

Fazekas Imre, Gábor András

JAVA PÉLDATÁR

mobiDIÁK könyvtár

Fazekas Imre, Gábor András

JAVA PÉLDATÁR

mobiDIÁK könyvtár
SOROZATSZERKESZTŐ
Fazekas István

Fazekas Imre, Gábor András

PhD hallgatók

Debreceni Egyetem

Informatikai Intézet

JAVA PÉLDATÁR

Oktatási segédanyag

Első kiadás

mobiDIÁK könyvtár
Debreceni Egyetem

Lektor: Dr. Juhász István

Copyright © Fazekas Imre, Gábor András, 2003

Copyright © elektronikus közlés mobiDIÁK könyvtár, 2003

mobiDIÁK könyvtár

Debreceni Egyetem

Informatikai Intézet

4010 Debrecen, Pf. 12.

Hungary

<http://mobidiak.inf.unideb.hu/>

A mű egyéni tanulmányozás céljára szabadon letölthető.

Minden egyéb felhasználás csak a szerző előzetes írásbeli engedélyével történhet.

A mű „A mobiDIÁK önszervező mobil portál” (IKTA, OMFB-00373/2003) és a

„GNU Iterátor, a legújabb generációs portál szoftver” (ITEM, 50/2003) projektek keretében készült.

Tartalomjegyzék

Tartalomjegyzék.....	6
1. Előszó.....	8
2. Az alkalmazástervezés alapjai.....	9
3. Egy minialkalmazási projekt.....	12
1. Példa	12
1.2. Példa	13
1.3. Példa	14
2. Példa	14
2.1. Példa	15
2.2. Példa	16
2.3. Példa	16
2.4. Példa	18
3. Példa	19
3.1. Példa	20
3.2. Példa	20
3.3. Példa	20
3.4. Példa	21
3.5. Példa	21
3.6. Példa	22
4. Példa	25
4.1. Példa	25
5. Példa	26
5.1. Példa	28
5.2. Példa	28
6. Példa	29
6.1. Példa	30
7. Példa	31
7.2. Példa	33
7.3. Példa	33
7.4. Példa	35
7.4.1. Példa	35
7.4.2. Példa	36
7.4.3. Példa	37
7.4.4. Példa	40
8. Példa	41
Feladatok	42
Egyéb feladatok.....	42
4. Zárthelyi dolgozatok példamegoldásai.....	47
2000/01 I. félév	47
1. Feladat	47
2. Feladat	49

2002/03 I. félév	50
1. Feladatsor	50
1. Feladat	50
2. Feladat	51
2. Feladatsor	53
1. Feladat	53
2. Feladat (módosított változat).....	54
3. Feladat	59
3. Feladatsor	66
1. Feladat	66
2. Feladat	70
3. Feladat	74
4. Feladat (módosított változat).....	75
2003/04 I. félév	80
1. Feladatsor	80
1. Feladat	80
2. Feladat	83
2. Feladatsor	85
1. Feladat	85
2. Feladat	86
3. Feladat	89
3. Feladatsor	95
1. Feladat	95
2. Feladat	97
3. Feladat	99
Ajánlott irodalom, hivatkozások.....	103

1. Előszó

Ez a jegyzet a Debreceni Egyetem Informatikai Intézetében az informatika tanár, programozó matematikus, programtervező matematikus szakok kötelező tárgyai között szereplő Programozás 2 tantárgy gyakorlataihoz készült. A gyakorlaton a kötelező nyelv a Java. Ezen nyelv programozási eszközeinek használatához, az objektumorientált paradigma gondolkodás- és alkalmazásmódjához nyújt segítséget a jegyzet. Két fő részből áll. Az első felében egy esettanulmány keretei között mutatja be a Java nyelv használatát, az alapvető programozási fogásokat. A második rész pedig az oktatásban az elmúlt tanévekben a számonkérésnél alkalmazott feladatsorok megoldását tartalmazza.

2. Az alkalmazástervezés alapjai

Alkalmazásunk fejlesztéséhez szükségünk van némi modellezői tudásra, hogy ezáltal rálátást nyerjünk a program egészére, összefüggéseiben jól lássuk az osztályok struktúráját, viselkedéseit, kapcsolatait. A vállalati szintű alkalmazásfejlesztés elméletével és gyakorlatával a **Rendszerfejlesztés technológiája** című tantárgy foglalkozik, nekünk kis alkalmazásunkhoz elegendő az ún. osztálydiagram megismerése. Az alkalmazás fejlesztését négy fő részre oszthatjuk fel: elemzés, tervezés, implementálás, tesztelés. A lépések sorrendje fontossági sorrendjüket is mutatja.

Az elemzés során a cél a probléma megismerése annak minden részletében. Ez az információgyűjtés, összefüggések, kapcsolatok feltárását, ok-okozati viszonyok megismerését jelenti. Ez a lépés független az informatikától, programozási nyelvektől, kódoktól.

A modellezés során megalkotjuk a rendszerünk *absztrakt változatát*, vagyis:

- elkészítjük az alkalmazás viselkedésének, felépítettségének, felhasználói felületeinek tervezetét,
- megtervezzük a rendszer elemeit, azok felépítését, viselkedését, kapcsolatait
- megtervezzük a rendszer folyamatainak sorrendiségét, időbeliségét
- stb.

Az implementálás a tervezés által előállított tervek valamilyen programozási nyelvre történő automatikus leképezése.

A tesztelés az elkészült alkalmazás tesztelési és valós körülmények közötti működtetése, annak stabilitási, működési hibáinak feltárása céljából.

Osztálydiagram

Fejlesztésünk során a tervezési eszközök egyik legfontosabb elemét: az osztálydiagramot fogjuk alkalmazni. Nyilván, az a szint, melyen itt alkalmazzuk messze áll a vállalati szintű alkalmazásfejlesztéstől, de az alapvető feladatokhoz éppen megfelelő lesz.

Az osztálydiagramon az OO alkalmazás fogalmait, kapcsolatait tervezzük meg. Osztályokat, interfészeket veszünk fel, megtervezzük belső felépítésüket, viselkedésüket, majd egymáshoz való viszonyaikat. A továbbiakban az alkalmazás vagy annak bizonyos elemei szemléltetésére használjuk.

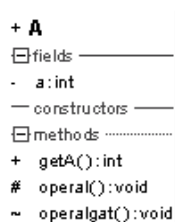
Minden osztály és interfész egy-egy „dobozként” jelenik meg.

Például:



A doboz első sora az osztály vagy interfész nevét jelzi, a többi része két illetve három részre van bontva. Osztályok esetén az adattagokat el kell határolni a viselkedésektől a jobb strukturáltság és átláthatóság kedvéért. Bizonyos tervező eszközök leválasztják a konstruktorokat a viselkedésektől és a kettő között, új szegmensben ábrázolják (jelen példában is.). Ez azzal magyarázható, hogy a konstruktorok valójában nem objektumviselkedést írnak le, hanem objektum létrehozása során a kezdő értékadás folyamatában játszanak meghatározó szerepet.

Az interfészeknél a két részre bontottság egyértelmű, hiszen csak *public final* adattagokat és *public abstract* metódusokat tartalmazhat.



Jelölő eszközök:

- „+” : public
- „-” : private
- „#” : protected
- „~” : csomagláthatóság
- „_” (aláhúzás) : static

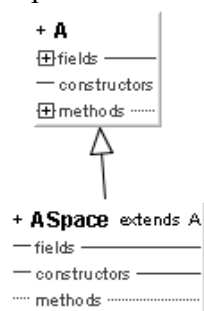
A csomagláthatóság jele erősen eltérhet tervező alkalmazásokon belül. Alkalmazásunkban konzekvensen a „~” jelet alkalmazzuk.

Minden egyes osztály esetén az összes direkt kapcsolatát ábrázolnunk kell:

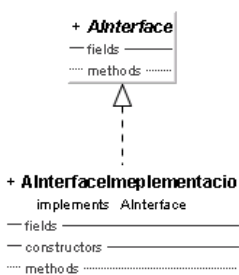
- *implementáció*: az osztály egy adott interfész implementációja
- *kiterjesztés*: az osztály egy adott osztály alosztálya
- *beágyazottság*: az osztály egy másik osztálynak belső osztálya
- *függőség*: az osztály függ egy másik osztálytól, vagyis hivatkozik rá, felhasználja, stb.
- *rész-egész viszony*: egy osztály egy másik osztály részét képezi, vagyis az osztályunk mint absztrakt típus, része a másik osztálynak mint absztrakt típusnak. Például:

Kormány-Kocsi

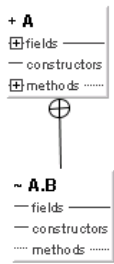
implementáció:



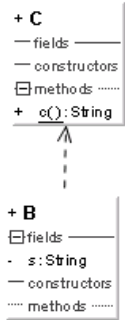
kiterjesztés:



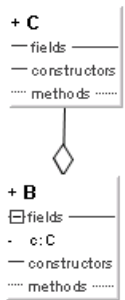
beágyazottság:



függőség:



rész-egész viszony:



Az ábrák elkészítéséhez bármilyen alkalmazásfejlesztési eszköz alkalmazható. Például: *Rational Rose*, *ArgoUML*, *PosseidonUML*, *MagicDraw UML*, *TogetherJ*, stb.

3. Egy minialkalmazási projekt

Alkalmazásunkban olyan példákat tárgyalunk, amelyek boltok tevékenységének nyilvántartását segítik. A példák fokozatosan vezetnek végig az Olvasót a különböző Java eszközök alkalmazási lehetőségein, a hangsúlyt a programozási fogásokra helyezve.

1. Példa : Modellezzük a következő osztályokat és implementáljuk ezeket:

- Elelmiszerbolt
- Tej

Lássuk milyen tulajdonságok és viselkedések reprezentálnak egy Tej típusú objektumot:

+ Tej
fields
- urtartalom: int
- gyarto: String
- szavatossagiIdo: Date
- zsirtartalom: double
- ar: long
constructors
+ Tej(urtartalom: int, gyarto: String, szavatossagiIdo: Date, zsirtartalom: double, ar: long)
methods
+ joMeg(): boolean
+ getUrtartalom(): int
+ getGyarto(): String
+ getSzavatossagiIdo(): Date
+ getZsirtartalom(): double
+ getAr(): long
+ toString(): String

Ehhez tartozó osztálydefiníció:

```
package bolt;

import java.util.Date;
public class Tej {

    private int urtartalom = 0; // ml-ben megadva
    private String gyarto;
    private Date szavatossagiIdo;
    private double zsirtartalom;
    private long ar;

    public Tej(int urtartalom, String gyarto, Date szavatossagiIdo, double
zsirtartalom, long ar) {
        this.urtartalom = urtartalom;
        this.gyarto = gyarto;
        this.szavatossagiIdo = szavatossagiIdo;
        this.zsirtartalom = zsirtartalom;
        this.ar = ar;
    }

    /* lekérdező metódusok */
    public boolean joMeg(){
        return szavatossagiIdo.before( new Date() );
    }
}
```

```

    public int getUrtartalom() {
        return urtartalom;
    }

    public String getGyarto() {
        return gyarto;
    }

    public Date getSzavatossagiIdo() {
        return szavatossagiIdo;
    }

    public double getZsirtartalom() {
        return zsirtartalom;
    }

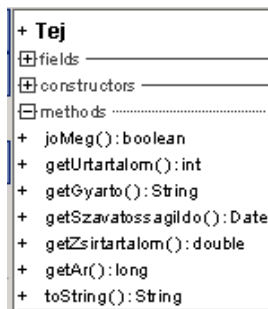
    public long getAr() {
        return ar;
    }
}

```

A csomagmegadás az alkalmazás részeit képező osztályok funkcionális összetartozását jelöli. Mivel az objektum egy kerek egészet jelent, így a külvilág számára az adattagok elfedésre kerültek, és lekérdező metódusokon keresztül érhetjük el azok tartalmát, létrehozva így az objektum tényleges identitását. Létrehozott osztályunkból a következő példányosítással lehet objektumot létrehozni:

```
Tej t = new Tej(1000,"Hegyek tej rt.",new Date(),2.8,121);
```

1.2. Példa: Amikor egy `Tej` objektumot meg akarunk jeleníteni a képernyőn, akkor a belső memóriacíme íródik ki, és nem azok az adatok, melyek rá, mint `Tej` típusú objektumra jellemzőek. Módosítsuk az osztályt úgy, hogy a sztring-reprezentációja is megfelelő legyen.



A `Tej` osztály új metódusa a következő:

```

...
    public String toString(){
        return "Gyártja:"+gyarto+" Szavatossági
idő:"+szavatossagiIdo+"Zsirtartalom:"+zsirtartalom;
    }
...

```

A Java ezzel a fejléccel rendelkező metódust keresi sztring-reprezentáció esetén. Így a

```
System.out.println( t );
```

a

```

Gyártja:Hegyek tej rt. Szavatossági idő:Mon Dec 01 12:16:57 CET 2003
Zsirtartalom:2.8

```

választ adja.

1.3. Példa: Valójában a zsírtartalom és az úrtartalom egy elég kis elemszámú halmazból vehetik fel értékeiket. Módosítsuk az osztályt, hogy a felhasználónak ne kelljen kitalálnia azt, hogy például az úrtartalom esetén ml-ben vagy dl-ben kell-e reprezentálni a kért értéket.

+ Tej
<input type="checkbox"/> fields
- final <u>LITER</u> :int
- final <u>FELLITER</u> :int
- final <u>POHAR</u> :int
- final <u>ZSIROS</u> :double
- final <u>FELZSIROS</u> :double
- urtartalom:int
- gyarto:String
- gyartasido:String
- szavatossagido:Date
- zsirtartalom:double
- ar:long
<input type="checkbox"/> constructors
<input type="checkbox"/> methods

Az új osztálydefiníció így nevesített konstansokkal bővült:

```
public class Tej {

    public static final int LITER = 1000;
    public static final int FELLITER = 500;
    public static final int POHAR = 200;

    public static final double ZSIROS = 2.8;
    public static final double FELZSIROS = 1.5;
    ...
}
```

Így a példányosítás sora is módosul:

```
Tej m = new Tej( Tej.LITER , "Hegyek tej rt.", new Date(),
Tej.ZSIROS, 121);
```

2. Példa: Most hozzuk létre a **Bolt** osztályt:

+ Bolt
<input type="checkbox"/> fields
- nev:String
- cim:String
- tulajdonos:String
- tejpult:Tej[]
- flag:int
<input type="checkbox"/> constructors
+ Bolt(nev:String, cim:String, tulajdonos:String, tejpult:Tej[])
<input type="checkbox"/> methods
+ getNev():String
+ getCim():String
+ getTulajdonos():String
+ vanMegTej():boolean
+ vasarolTej(m:Tej):Tej

A hozzá tartozó Java kód:

```
package bolt;
public class Bolt {

    private String nev, cim, tulajdonos;
    private Tej[] tejpult;
```

```

private int flag;

public Bolt(String nev, String cim, String tulajdonos, Tej[] tejpult) {
    this.nev = nev;
    this.cim = cim;
    this.tulajdonos = tulajdonos;
    this.tejpult = tejpult;
    flag=tejpult.length-1;
}

public String getNev() {
    return nev;
}

public String getCim() {
    return cim;
}




public String getTulajdonos() {
    return tulajdonos;
}

public boolean vanMegTej() {
    return flag>=0;
}

public Tej vasarolTej(Tej m){
    return tejpult[flag--];
}
}

```

2.1. Példa: Egy boltból nemcsak vásárolni lehet, hanem időnként fel is kell azt tölteni. Ennek megfelelően módosítsuk a `Bolt` osztályt.

+ Bolt
 fields
 constructors
+ Bolt(nev:String, cim:String, tulajdonos:String, tejpult:Tej[])
+ Bolt(nev:String, cim:String, tulajdonos:String)
 methods
+ getNev():String
+ getCim():String
+ getTulajdonos():String
+ vanMegTej():boolean
+ vasarolTej(m: Tej):Tej
+ feltoltTej(m: Tej):void

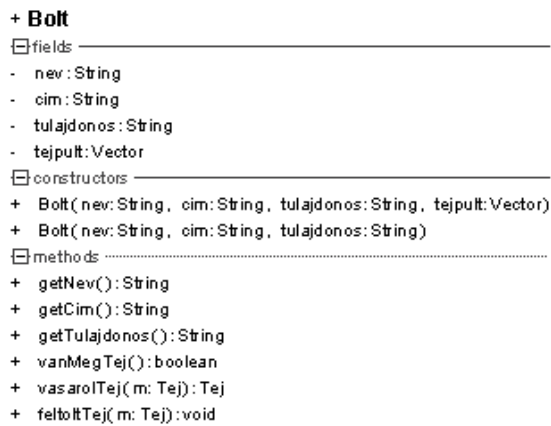
Ennek megfelelően az újonnan beszúrt sorok a következők:

```

...
    public Bolt(String nev, String cim, String tulajdonos) {
        this(nev, cim, tulajdonos, new Tej[100]);
    }
...
    public void feltoltTej(Tej m){
        tejpult[++flag] = m;
    }
...

```

2.2. Példa: A tejek mennyisége folyamatosan változik, ezért alkalmazzunk dinamikus adattárolást.



A módosított Java kódok:

```

public class Bolt {

    private String nev, cim, tulajdonos;
    private Vector tejpult;
    public Bolt(String nev, String cim, String tulajdonos, Vector tejpult)
    {
        this.nev = nev;
        this.cim = cim;
        this.tulajdonos = tulajdonos;
        this.tejpult = tejpult;
    }
    public Bolt(String nev, String cim, String tulajdonos) {
        this(nev, cim, tulajdonos, new Vector());
    }
    ...
    public boolean vanMegTej() {
        return tejpult.isEmpty();
    }

    public void feltoltTej(Tej m){
        tejpult.add( m) ;
    }

    public Tej vasarolTej(Tej m){
        return (Tej)tejpult.remove( tejpult.indexOf(m) );
    }
}
  
```

2.3. Példa: Vásárlás során sohasem azt mondjuk, hogy „ezt a két tej objektumot kérem” és átadjuk a tejek referenciáját. A mai boltok minden esetben vonalkóddal dolgoznak, így mind a feltöltés, mind a vásárlás alapja az áru kódja. Valósítsuk meg a vonalkód alapú tárolást.

Az osztálmódosításokat követő osztálydiagramok:

+ Tej <div>fields</div> <ul style="list-style-type: none"> - vonalkod: long - final LITER: int - final FELLITER: int - final POHAR: int - final ZSIROS: double - final FELZSIROS: double - urtartalom: int - gyarto: String - szavatossagiIdo: Date - zsirtartalom: double - ar: long <div>constructors</div> <ul style="list-style-type: none"> + Tej(vonalkod: long, urtartalom: int, gyarto: String, szavatossagiIdo: Date, zsirtartalom: double, ar: long) <div>methods</div> <ul style="list-style-type: none"> + getVonalkod(): long + joMeg(): boolean + getUrtartalom(): int + getGyarto(): String + getSzavatossagiIdo(): Date + getZsirtartalom(): double + getAr(): long + toString(): String 	+ Bolt <div>fields</div> <ul style="list-style-type: none"> - nev: String - cim: String - tulajdonos: String - tejpult: Hashtable <div>constructors</div> <ul style="list-style-type: none"> + Bolt(nev: String, cim: String, tulajdonos: String, tejpult: Hashtable) + Bolt(nev: String, cim: String, tulajdonos: String) <div>methods</div> <ul style="list-style-type: none"> + getNev(): String + getCim(): String + getTulajdonos(): String + vanMegTej(): boolean + vasarolTej(vonalkod: long): Tej + feltoltTej(m: Tej): void
--	--

Módosított Java kódok a Tej osztályban:

```
...
public class Tej {
    private long vonalkod;
    ...
    public Tej(long vonalkod, int urtartalom, String gyarto, Date
szavatossagiIdo, double
    zsirtartalom, long ar) {
        this.vonalkod = vonalkod;
        this.urtartalom = urtartalom;
        this.gyarto = gyarto;
        this.szavatossagiIdo = szavatossagiIdo;
        this.zsirtartalom = zsirtartalom;
        this.ar = ar;
    }
    public long getVonalkod() {
        return vonalkod;
    }
    ...
}
```

Módosított Java kódok a Bolt osztályban:

```
...
public class Bolt {
    private Hashtable tejpult;
    public Bolt(String nev, String cim, String tulajdonos) {
        this(nev, cim, tulajdonos, new Hashtable() );
    }

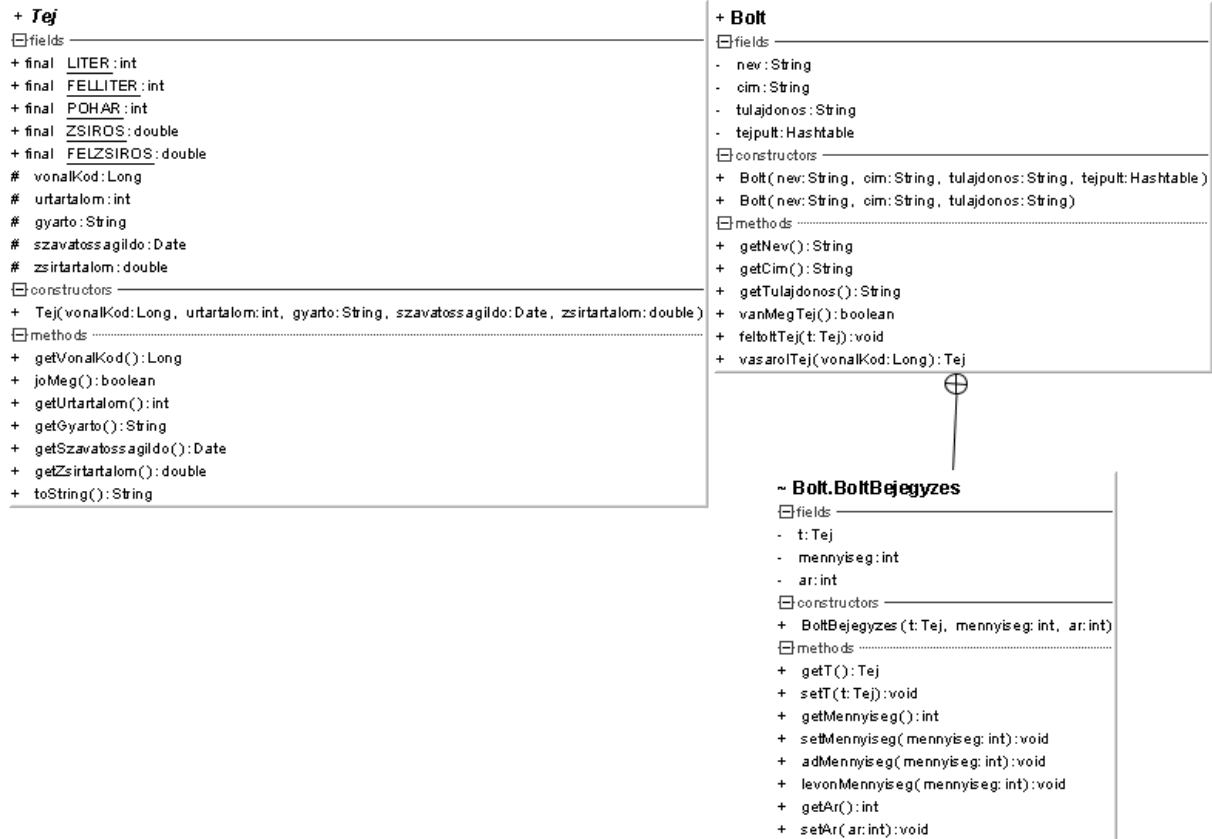
    public Bolt(String nev, String cim, String tulajdonos, Hashtable
tejpult) {
        this.nev = nev;
        this.cim = cim;
        this.tulajdonos = tulajdonos;
        this.tejpult = tejpult;
    }
    ...
    public void feltoltTej(Tej m){
        tejpult.put( new Long( m.getVonalkod() ), m );
    }

    public Tej vasarolTej(long vonalkod){
        return (Tej)tejpult.remove( tejpult.get( new Long( vonalkod ) ) );
    }
}
```

```
}
...
```

2.4. Példa: Egy bolt nem minden árut külön-külön tart nyilván, hanem minden árutípushoz egy-egy bejegyzést rendel, mely azonosítja az árut, tartalmazza az árat, a raktáron lévő mennyiséget. Tehát az ár nem az áru sajátossága, hanem a bolté. Módosítsuk a `Bolt` osztályt, hogy ilyen bejegyzéseket kezeljen.

Módosított osztályok osztálydiagramja:



Jól láthatóan az ár adattag eltűnt a `Tej` osztályból.

A `Bolt` osztály új sorai:

```
public class Bolt {
    ...
    class BoltBejegyzes{
        private Tej t;
        private int mennyiseg;
        private int ar;
        public BoltBejegyzes(Tej t, int mennyiseg, int ar) {
            this.t = t;
            this.mennyiseg = mennyiseg;
            this.ar = ar;
        }
        public Tej getT() {
            return t;
        }
        public void setM(Tej m) {
            this.t = t;
        }
        public int getMennyiseg() {
            return mennyiseg;
        }
    }
}
```

```

    }
    public void setMennyiseg(int mennyiseg) {
        this.mennyiseg = mennyiseg;
    }
    public void adMennyiseg(int mennyiseg) {
        this.mennyiseg += mennyiseg;
    }
    public void levonMennyiseg(int mennyiseg) {
        this.mennyiseg -= mennyiseg;
    }
    public int getAr() {
        return ar;
    }
    public void setAr(int ar) {
        this.ar = ar;
    }
}

```

A Bolt osztály módosított sorai:

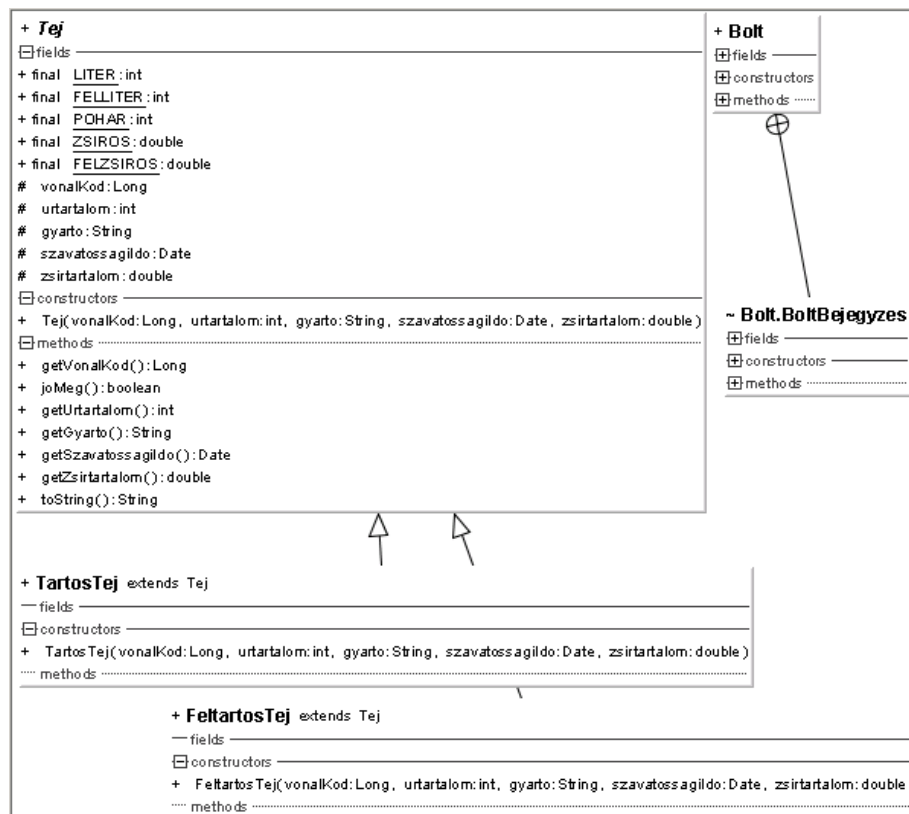
```

...
    public void feltoltTej(Tej t){
        BoltBejegyzes b = (BoltBejegyzes) tejpult.get( m.getVonalKod() );
        if( b == null){
            b = new BoltBejegyzes(t, 1, 100);
            tejpult.put( t.getVonalKod(), b );
        }
        else b.adMennyiseg( 1 );
    }

    public Tej vasarolTej(Long vonalKod){
        BoltBejegyzes b = (BoltBejegyzes) tejpult.get( vonalKod );
        if( b != null){
            b.levonMennyiseg( 1 );
            return b.getT();
        }
        return null;
    }
    ...

```

3. Példa: Egy bolt nem csak egyfajta tejet árul, hanem tartósat, féltartósat, stb. Módosítsuk úgy alkalmazásunkat, hogy ezen fogalmak megjelenjenek.



3.1. Példa: A `Tej` egy absztrakt fogalom, vagyis olyan típust jelöl, mely inkább gyűjtőnevet jelöl és nem konkrétat. Ennek megfelelően módosítsuk az alkalmazást.

```

...
public abstract class Tej {
...

```

Mivel egy bolt szempontjából mindegy, hogy egy tej milyen, csak egyszerűen tejet akarnak árulni, a `BoltBejegyzes` osztály nem változik. Így általános típusú objektumokat tárolunk, nem kell a specializációval foglalkoznunk.

3.2. Példa: Mivel a `TartosTej` és a `FeltartosTej` egyazon fogalom kiterjesztései, ennek a csomagspecifikációban is tükröződnie kellene. Módosítsuk ezen osztályokat.

Módosított osztályok:

```

package bolt.tej;
import bolt.Tej;
import java.util.Date;
public class FeltartosTej extends Tej{ ...

package bolt.tej;
import bolt.Tej;
import java.util.Date;
public class TartosTej extends Tej{ ...

```

3.3 Példa: Az új absztrakciós szint bonyolította alkalmazásunk használatát, hiszen egy új tej létrehozásához minden konkrét alosztályt ismerni kellene elérési úttal együtt. Ezért vegyünk

fel egy absztrakt osztályt és benne kiszolgáló metódusokat, melyek konkrét tej alosztályok példányait hozzák létre.

```

+ TejFactory
- fields -----
- constructors -----
- methods -----
+ ujTartosTej(vonalkod:Long, urtartalom:int, gyarto:String, szavatossagido:Date, zsirtartalom:double, ar:long):Tej
+ ujFeltartosTej(vonalkod:Long, urtartalom:int, gyarto:String, szavatossagido:Date, zsirtartalom:double, ar:long):Tej

```

A `TejFactory` statikus metódusokkal rendelkezik, hiszen a példányosítás egy szolgáltatás, mellyel a tej típusok készítése könnyebbé válik. Például:

```
Tej tartostej = TejFactory.ujTartosTej( ... );
```

Így nem kell sem osztálynevet, sem konstruktorokat, sem inicializátorokat, sem csomagneveket ismernünk.

3.4 Példa: Vegyünk fel a `Sajt` osztályt az alkalmazásba.

```

+ Sajt
- fields -----
# suly:double
# zsirtartalom:double
# vonalkod:Long
# gyarto:String
# szavatossagido:Date
- constructors -----
+ Sajt(vonalkod:Long, suly:double, gyarto:String, szavatossagido:Date, zsirtartalom:double)
- methods -----
+ getSuly():double
+ getZsirtartalom():double
+ toString():String
+ joMeg():boolean
+ getSztatossagido():Date
+ getVonalkod():Long
+ getGyarto():String

```

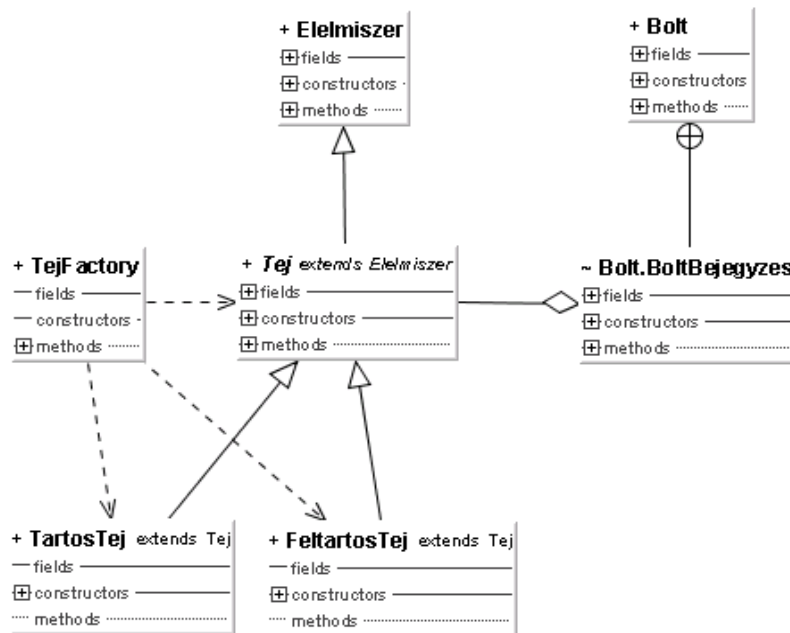
3.5 Példa: Bővítsük projektünket egy `Elelmiszer` osztállyal. Gondoljuk végig, hogy az élelmiszer milyen fogalmat jelöl!

```

+ Elelmiszer
- fields -----
# vonalkod:Long
# gyarto:String
# szavatossagido:Date
- constructors -----
+ Elelmiszer(vonalkod:Long, gyarto:String, szavatossagido:Date)
- methods -----
+ getVonalkod():Long
+ joMeg():boolean
+ getGyarto():String
+ getSztatossagido():Date
+ toString():String

```

Az `Elelmiszer` gyűjtőnév jellege miatt absztrakt, és minden tej és sajt ősosztálya is, mivel azok általános esetét jelenti. Így a létrejövő hierarchia:



Így az élelmiszerekre jellemző tulajdonságok és viselkedések öröklődéssel kerülnek a tej és sajttípusokhoz.

```

public abstract class Tej extends Elelmiszer{
    ...
    public Tej(Long vonalKod, int urtartalom, String gyarto, Date
szavatossagiIdo, double zsirtartalom) {
        super(vonalKod, gyarto, szavatossagiIdo);
        this.urtartalom = urtartalom;
        this.zsirtartalom = zsirtartalom;
    }
    ...
    public String toString(){
        return super.toString()+"Zsirtartalom:"+zsirtartalom;
    }
    ...
}

public class Sajt extends Elelmiszer{
    ...
    public Sajt(Long vonalKod, double suly, String gyarto, Date
szavatossagiIdo, double zsirtartalom) {
        super(vonalKod, gyarto, szavatossagiIdo);
        this.suly = suly;
        this.zsirtartalom = zsirtartalom;
    }
    ...
    public String toString(){
        return super.toString()+"Zsirtartalom:"+zsirtartalom;
    }
}

```

3.6 Példa: Egy bolt számára mindegy, hogy tej, vagy sajt objektumokat tárol az adatbázisban. Egyszerűsítsük ezen élelmiszerek kezelését, könnyítsük meg az adatbázis módosítását segédmetódusokkal.

+ Bolt <div>fields</div> <ul style="list-style-type: none"> - nev:String - tulajdonos:String - cim:String - elelmiszerpult:Hashtable <div>constructors</div> <ul style="list-style-type: none"> + Bolt(nev:String, tulajdonos:String, cim:String) + Bolt(nev:String, tulajdonos:String, cim:String, elelmiszerpult:Hashtable) <div>methods</div> <ul style="list-style-type: none"> + getNev():String + getTulajdonos():String + getCim():String - vanMegAdottAru(c:Class):boolean + vanMegTej():boolean + vanMegSajt():boolean + feltoitElelmiszerrel(vonalkod:Long, mennyiseg:long):void + feltoitUjElelmiszerrel(e:Elelmiszer, mennyiseg:long, ar:long):void + torolElelmiszert(vonalkod:Long):void + vasarolElelmiszert(vonalkod:Long, mennyiseg:long):void 	+ Bolt.BoltBejegyzes <div>fields</div> <ul style="list-style-type: none"> - e:Elelmiszer - mennyiseg:long - ar:long <div>constructors</div> <ul style="list-style-type: none"> + BoltBejegyzes(e:Elelmiszer, mennyiseg:long, ar:long) <div>methods</div> <ul style="list-style-type: none"> + getElelmiszer():Elelmiszer + setElelmiszer(e:Elelmiszer):void + getMennyiseg():long + setMennyiseg(mennyiseg:long):void + adMennyiseg(mennyiseg:long):void + levonMennyiseg(mennyiseg:long):void + getAr():long + setAr(ar:int):void
---	--

Az újonnan megjelenő publikus kiszolgáló módszerek implementációja:

```
package bolt;

import java.util.Hashtable;
import java.util.Enumuration;

public class Bolt {

    private String nev, tulajdonos, cim;
    private Hashtable elelmiszerpult;

    public class BoltBejegyzes{
        private Elelmiszer e;
        private long mennyiseg;
        private long ar;
        public BoltBejegyzes(Elelmiszer e, long mennyiseg, long ar) {
            this.e = e;
            this.mennyiseg = mennyiseg;
            this.ar = ar;
        }
        public Elelmiszer getElelmiszer() {
            return e;
        }
        public void setElelmiszer(Elelmiszer e) {
            this.e = e;
        }
        public long getMennyiseg() {
            return mennyiseg;
        }
        public void setMennyiseg(long mennyiseg) {
            this.mennyiseg = mennyiseg;
        }
        public void adMennyiseg(long mennyiseg) {
            this.mennyiseg += mennyiseg;
        }
        public void levonMennyiseg(long mennyiseg) {
            this.mennyiseg -= mennyiseg;
        }
        public long getAr() {
```

```

        return ar;
    }
    public void setAr(int ar) {
        this.ar = ar;
    }
}

public Bolt(String nev, String tulajdonos, String cim) {
    this(nev, tulajdonos, cim, new Hashtable());
}

public Bolt(String nev, String tulajdonos, String cim, Hashtable
elelmiszerpult) {
    this.nev = nev;
    this.tulajdonos = tulajdonos;
    this.cim = cim;
    this.elelmiszerpult = elelmiszerpult;
}

public String getNev() {
    return nev;
}

public String getTulajdonos() {
    return tulajdonos;
}

public String getCim() {
    return cim;
}

private boolean vanMegAdottAru(Class c){
    for( Enumeration e = elelmiszerpult.elements();
e.hasMoreElements();){
        BoltBejegyzes b = (BoltBejegyzes)e.nextElement();
        if( c.isInstance( b.getElelmiszer() ) && b.getMennyiseg()>0 )
return true;
    }
    return false;
}

public boolean vanMegTej(){
    return vanMegAdottAru( Tej.class );
}

public boolean vanMegSajt(){
    return vanMegAdottAru( Sajt.class );
}

public void feltoltElelmiszerrel(Long vonalKod, long mennyiseg){
    BoltBejegyzes b = (BoltBejegyzes) elelmiszerpult.get( vonalKod );
    b.adMennyiseg( mennyiseg );
}

public void feltoltUjElelmiszerrel(Elelmiszer e, long mennyiseg, long
ar){
    BoltBejegyzes b = new BoltBejegyzes(e, mennyiseg, ar);
    b.adMennyiseg( mennyiseg );
}

public void torolElelmiszert(Long vonalKod){

```



```

        elelmiszerpult.remove( vonalKod );
    }

    public void vasarolElelmiszert(Long vonalKod, long mennyiseg){
        BoltBejegyzes b = (BoltBejegyzes) elelmiszerpult.get( vonalKod );
        if( b != null)
            b.levonMennyiseg( mennyiseg );
    }
}

```

4. Példa: Készítsük fel alkalmazásunkat az esetlegesen bekövetkező hibák kezelésére is. Törekedjünk a konzisztens működésre.

Új kivételosztályaink:

+ NemLetezoAruKivetel extends Exception <hr/> fields <hr/> constructors <hr/> + NemLetezoAruKivetel(message: String) <hr/> methods	+ TulSokLevonasKivetel extends Exception <hr/> fields <hr/> constructors <hr/> + TulSokLevonasKivetel(message: String) <hr/> methods
--	--

Nézzük a Bolt osztály módosult metódusait:

```

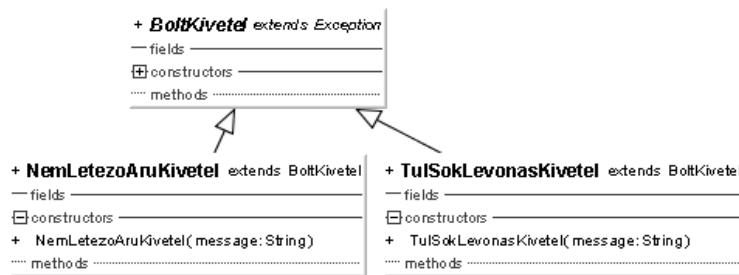
...
    public void feltoltElelmiszerral(Long vonalKod, long mennyiseg) throws
NemLetezoAruKivetel{
        BoltBejegyzes b = (BoltBejegyzes) elelmiszerpult.get( vonalKod );
        if( b == null) throw new NemLetezoAruKivetel("Ilyen aru
nincsen:"+vonalkod);
        b.adMennyiseg( mennyiseg );
    }

    public void torolElelmiszert(Long vonalKod) throws NemLetezoAruKivetel{
        if( elelmiszerpult.remove( vonalKod ) == null ) throw new
NemLetezoAruKivetel("Ilyen aru nincsen:"+vonalkod);
    }

    public void vasarolElelmiszert(Long vonalKod, long mennyiseg) throws
NemLetezoAruKivetel, TulSokLevonasKivetel{
        BoltBejegyzes b = (BoltBejegyzes) elelmiszerpult.get( vonalKod );
        if( b == null) throw new NemLetezoAruKivetel("Ilyen aru
nincsen:"+vonalkod);
        if( b != null){
            if( b.getMennyiseg() < mennyiseg ) throw new
TulSokLevonasKivetel("Nincs elegendo mennyiseg:"+vonalkod);
            b.levonMennyiseg( mennyiseg );
        }
    }
...

```

4.1. Példa: Egy alkalmazás során sok saját kivételtípust kezelhetünk. Ilyenkor a metódusok specifikációjánál „tobzódás” léphet fel, így annak hívásánál is körülményes az események kezelése. Egyszerűsítsünk az alkalmazás kivételmodelljén:



A BoltKivétel a bolti alkalmazásban fellépő kivételek gyűjtőneve.

A Bolt osztályban módosított metódusok:

```

...
public void feltoltElelmiszerral(Long vonalKod, long mennyiseg) throws
BoltKivétel{
...

```

```

public void vasarolElelmiszert(Long vonalKod, long mennyiseg) throws
BoltKivétel{
...

```

```

public void torolElelmiszert(Long vonalKod) throws BoltKivétel{
...

```

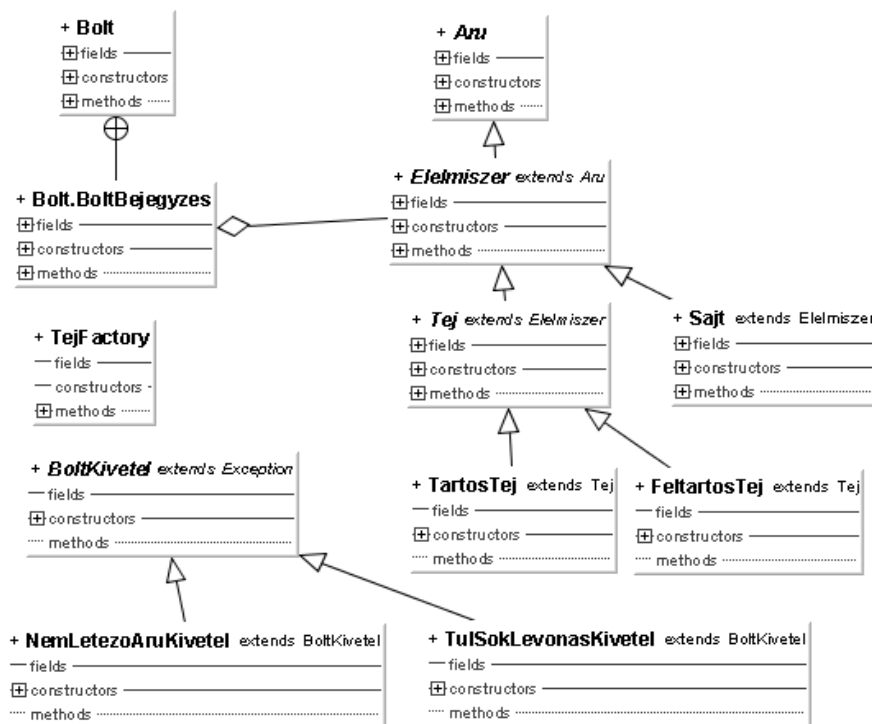
Így a kivételkezelő egyszerűsített alkalmazása:

```

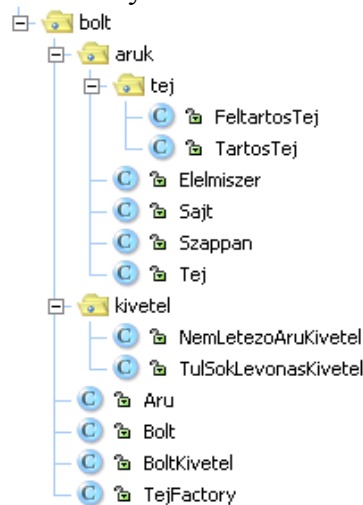
try{
    b.vasarolElelmiszert(1111111,12);
}
catch(BoltKivétel b){ b.printStackTrace(); ... }

```

5. Példa: Modellünket módosítsuk úgy, hogy szappant is áruljon a boltunk.



Bevezetjük az absztrakt áru fogalmát, mely a szappan és az élelmiszer gyűjtőneve is egyben.
Az osztálystruktúra módosított felépítése:



A Szappan osztály definíciója:

```

package bolt.aruk;
import bolt.Aru;
public class Szappan extends Aru{

    public static final char AMOSOHATAS = 'A';
    public static final char BMOSOHATAS = 'B';
    private final char mosohatas;

    public Szappan(Long vonalKod, String gyarto, char mosohatas) {
        super(vonalKod, gyarto);
        this.mosohatas = mosohatas;
    }

}
  
```

Az Elelmiszer módosított osztálya:

```

package bolt.aruk;
import bolt.Aru;
import java.util.Date;
public abstract class Elelmiszer extends Aru{

    protected Date szavatossagiIdo;

    public Elelmiszer(Long vonalKod, String gyarto, Date szavatossagiIdo) {
        super(vonalKod, gyarto);
        this.szavatossagiIdo = szavatossagiIdo;
    }

    public Date getSzavatossagiIdo() {
        return szavatossagiIdo;
    }

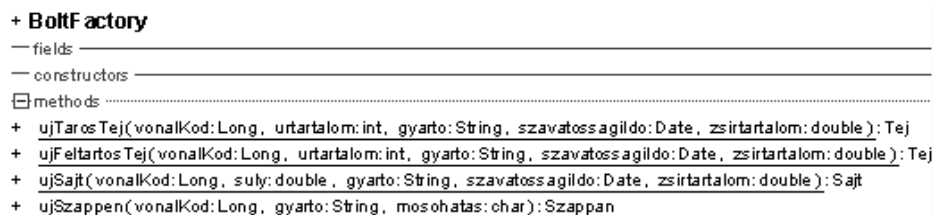
    public boolean joMeg(){
        return new Date().before( szavatossagiIdo );
    }

}
  
```

Módosítást kellett végezni a `Bolt` osztályon is:

```
public class Bolt {
    ...
    private Hashtable arupult;
    ...
    public class BoltBejegyzes{
        private Aru a;
        ...
        public BoltBejegyzes(Aru a, long mennyiseg, long ar) {
            this.a = a;
            this.mennyiseg = mennyiseg;
            this.ar = ar;
        }
        public Aru getAru() {
            return a;
        }
        public void setAru(Aru a) {
            this.a = a;
        }
        ...
    }
    ...
}
```

5.1. Példa: `TejFactory` kiszolgáló osztályunk elég „kiszolgált” már. Módosítsuk az újonnan elkészült bolti alkalmazás alapján.



```
+ BoltFactory
- fields
- constructors
+ methods
+ ujTaros(Tej(vonalkod:Long, irtartalom:int, gyarto:String, szavatossagido:Date, zsirtartalom:double)):Tej
+ ujFeltartos(Tej(vonalkod:Long, irtartalom:int, gyarto:String, szavatossagido:Date, zsirtartalom:double)):Tej
+ ujSzajtt(vonalkod:Long, suly:double, gyarto:String, szavatossagido:Date, zsirtartalom:double):Szajtt
+ ujSzajtt(vonalkod:Long, gyarto:String, mosohatas:char):Szajtt
```

Ez az osztály minden `Aru` típus példányosításához segítséget nyújt.

5.2. Példa: A példányosítások során a sok plusz munkát az olyan paraméterek megadása jelenti, mely értéke nagy valószínűség szerint minden esetben megegyezik. Adjunk ennek segítésére módszereket.

```

+ BoltFactory
- fields
- constructors
+ methods
+ ujTarosTej(vonalKod:Long, urtartalom:int, gyarto:String, szavatossagido:Date, zsirtartalom:double):Tej
+ ujFeltartosTej(vonalKod:Long, urtartalom:int, gyarto:String, szavatossagido:Date, zsirtartalom:double):Tej
+ ujFelzsirosTarosTej(vonalKod:Long, urtartalom:int, gyarto:String, szavatossagido:Date):Tej
+ ujZsirosTarosTej(vonalKod:Long, urtartalom:int, gyarto:String, szavatossagido:Date):Tej
+ ujFelzsirosFeltartosTej(vonalKod:Long, urtartalom:int, gyarto:String, szavatossagido:Date):Tej
+ ujZsirosFeltartosTej(vonalKod:Long, urtartalom:int, gyarto:String, szavatossagido:Date):Tej
+ ujFelzsirosLiteresTarosTej(vonalKod:Long, gyarto:String, szavatossagido:Date):Tej
+ ujZsirosLiteresTarosTej(vonalKod:Long, gyarto:String, szavatossagido:Date):Tej
+ ujFelzsirosLiteresFeltartosTej(vonalKod:Long, gyarto:String, szavatossagido:Date):Tej
+ ujZsirosLiteresFeltartosTej(vonalKod:Long, gyarto:String, szavatossagido:Date):Tej
+ ujSajt(vonalKod:Long, suly:double, gyarto:String, szavatossagido:Date, zsirtartalom:double):Sajt
+ ujSzappen(vonalKod:Long, gyarto:String, mosohatas:char):Szappen
+ ujAMosohatasuSzappen(vonalKod:Long, gyarto:String):Szappen

```

```

package bolt;

import bolt.aruk.Tej;
import bolt.aruk.Sajt;
import bolt.aruk.Szappen;
import bolt.aruk.tej.TartosTej;
import bolt.aruk.tej.FeltartosTej;

import java.util.Date;

public class BoltFactory {

    public static Tej ujTarosTej(Long vonalKod, int urtartalom, String
gyarto, Date
szavatossagiIdo, double zsirtartalom){
        return new TartosTej( vonalKod, urtartalom, gyarto, szavatossagiIdo,
zsirtartalom);
    }

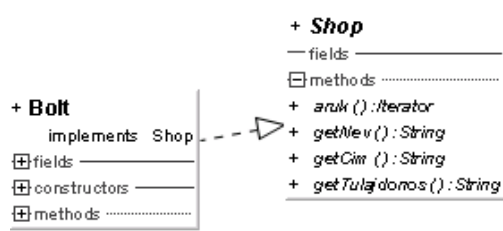
    public static Tej ujFelzsirosTarosTej(Long vonalKod, int urtartalom,
String gyarto, Date
szavatossagiIdo){
        return new TartosTej( vonalKod, urtartalom, gyarto, szavatossagiIdo,
Tej.FELZSIROS);
    }

    public static Tej ujFelzsirosLiteresTarosTej(Long vonalKod, String
gyarto, Date
szavatossagiIdo){
        return new TartosTej( vonalKod, Tej.LITER, gyarto, szavatossagiIdo,
Tej.FELZSIROS);
    }
    ...
}

```

Ezen metódusokkal nagymértékben elősegíthetjük egy későbbi felhasználói interfész integrálását.

6. Példa: Vegyünk a következő fogalmat: írható CD, mely élelmiszerboltban (ma még általában) nem kapható. Alkossuk meg a Shop viselkedést, mely egy eladási viselkedés OO reprezentációja.



A `Shop` interfészben egy eladással foglalkozó objektum legfontosabb viselkedéseit definiáltuk. Az interfész kódja:

```

package bolt;

import java.util.Iterator;

public interface Shop {

    Iterator aruk();
    String getNev();
    String getCim();
    String getTulajdonos();

}
  
```

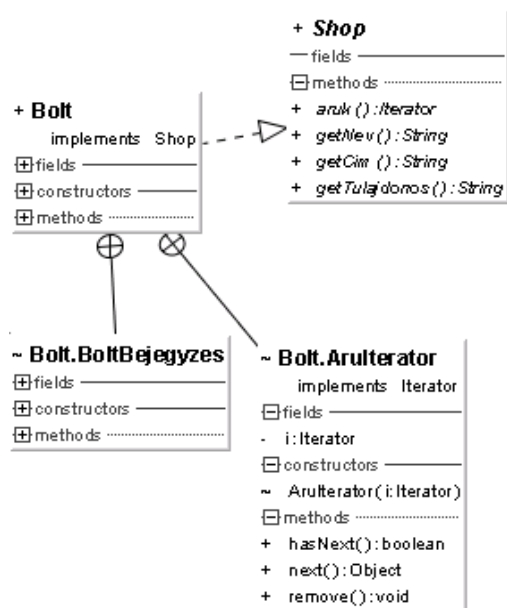
A `Bolt` osztály módosított változata:

```

package bolt;

...
public class Bolt implements Shop{
    ...
    public Iterator aruk() {
        return arupult.values().iterator();
    }
    ...
}
  
```

6.1. Példa: Mivel az árú ezen típusú iterációja a `Hashtable` belső implementációját használja, kiadja minden objektum referenciáját. Írjunk saját iterátor implementációt, mely csak az árú vonalkódját adja ki.



A Bolt osztály módosított változata:

```

package bolt;
...
public class Bolt implements Shop{
    ...
    class AruIterator implements Iterator{

        private Iterator i;

        AruIterator(Iterator i){
            this.i = i;
        }

        public boolean hasNext() {
            return i.hasNext();
        }

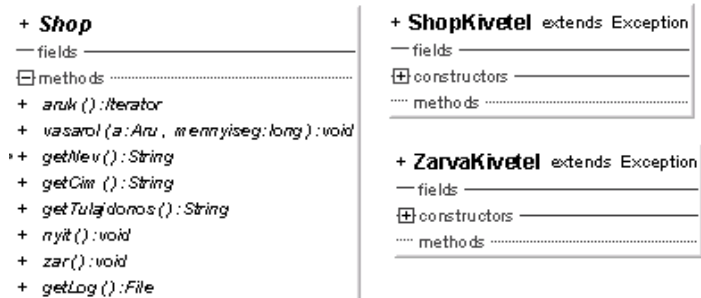
        public Object next() {
            return ((Aru) i.next()).getVonalKod();
        }

        public void remove() {
            i.remove();
        }

    }

    public Iterator aruk() {
        return new AruIterator(arupult.values().iterator());
    }
    ...
}
  
```

7. Példa: Minden `Shop` objektum rendelkezik nyitás és zárás viselkedéssel. Amikor a `Shop` zárva van, semmilyen művelet nem végezhető vele. Módosítsuk ennek megfelelően a definíciókat.



A bolt zárása utáni árulekérdezést vagy vásárlást a bolt kivétel dobásával jelzi. Lássuk a módosított osztályokat:

```

Shop:
package bolt;

import java.util.Iterator;

public interface Shop {

    Iterator aruk() throws ZarvaKivétel;
    void vasarol(Aru a, long mennyiség) throws ZarvaKivétel, BoltKivétel;
    String getNev();
    String getCim();
    String getTulajdonos();
    void nyit();
    void zar();
}
  
```

```

Bolt:
package bolt;

...
public class Bolt implements Shop{
    private boolean nyitva=false;
    ...
    class AruIterator implements Iterator{

        private Iterator i;

        AruIterator(Iterator i){
            this.i = i;
        }

        public boolean hasNext() {
            return i.hasNext();
        }

        public Object next() {
            return ((Aru) (i.next())).getVonalKod();
        }

        public void remove() {
            i.remove();
        }

    }
}
  
```



```

    public Iterator aruk() throws ZarvaKivetel{
        if(!nyitva) throw new ZarvaKivetel("A bolt zárva van");
        return new AruIterator(arupult.values().iterator());
    }

    public void vasarol(Aru a, long mennyiseg) throws ZarvaKivetel,
    BoltKivetel {
        if(!nyitva) throw new ZarvaKivetel("A bolt zárva van");
        vasarolElelmiszert(a.getVonalKod(),mennyiseg);
    }

    public String getNev() {
        return nev;
    }

    public String getTulajdonos() {
        return tulajdonos;
    }

    public void nyit() {
        nyitva = true;
    }

    public void zar() {
        nyitva = false;
    }
    ...
}

```

7.2. Példa: Mivel vásárolni az iterátor által adott `Aru` típusú objektumok alapján lehet, a belső iterátor reprezentációt is módosítani kell, hogy `Aru` típusú legyen, de mégis elfedje a belső reprezentációt. Erre használjuk a névtelen példányosítást!

Az iterátor osztály módosításának eredménye:

```

class AruIterator implements Iterator{
    ...
    public Object next() {
        Aru a = (Aru)i.next();
        return new Aru( a.getVonalKod(), a.getGyarto()){};
    }
    ...
}

```

7.3. Példa: Naplózza minden `Shop` nyitva tartása idején folyamatosan tevékenységeit egy állományba, melyet zárást követően le is lehessen kérdezni!

+ Shop — fields — ☐ methods + <i>aruk()</i> : <i>Iterator</i> + <i>vasarol(a:Aru, mennyiseg:long)</i> : <i>void</i> + <i>getNev()</i> : <i>String</i> + <i>getCim()</i> : <i>String</i> + <i>getTulajdonos()</i> : <i>String</i> + <i>nyit()</i> : <i>void</i> + <i>zar()</i> : <i>void</i> + <i>getLog()</i> : <i>File</i>	+ ShopKivetel extends <i>Exception</i> — fields — ☐ constructors — methods + ZavarKivetel extends <i>Exception</i> — fields — ☐ constructors — methods
--	---

Módosított definíciók:

Shop:

```
...
public interface Shop {
    ...
    File getLog() throws ShopKivetel;
}
```

Bolt:

```
...
public class Bolt implements Shop{
    ...
    public void nyit() {
        nyitva = true;
        try {
            logFile = new File( logPath+File.separator+"Bolt"+new Date() );
            logWriter = new FileWriter( logFile );
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void zar() {
        nyitva = false;
        try {
            logWriter.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public File getLog() throws ShopKivetel{
        if(!nyitva) return logFile;
        else throw new ShopKivetel("a bolt nincs zárva");
    }

    private void log(String s){
        try {
            logWriter.write(s);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private boolean vanMegAdottAru(Class c){
        log("Árutípus készletezettségének ellenőrzése:"+c);
    }
}
```

```

        ...
    }

    public void feltoltElelmiszerrel(Long vonalKod, long mennyiseg) throws
    BoltKivetel{
        log("Elelmiszerfeltoltes a"+vonalKod+" arubol "+mennyiseg+"
mennyiseggel");
        ...
    }

    public void feltoltUjElelmiszerrel(Elelmiszer e, long mennyiseg, long
ar){
        log("Új élelmiszer feltoltese:"+e+" "+mennyiseg+" mennyiseggel");
        ...
    }

    public void torolElelmiszert(Long vonalKod) throws BoltKivetel{
        log("Elelmiszertorles:"+vonalKod);
        ...
    }

    public void vasarolElelmiszert(Long vonalKod, long mennyiseg) throws
    BoltKivetel{
        log("Elelmiszervásárlás a"+vonalKod+" arubol "+mennyiseg+"
mennyiseggel");
        ...
    }
    ...
}

```

7.4. Példa: Mivel a napló feldolgozása sokrétű lehet, interfészekkel kell segíteni a rendszergazda információgyűjtésének folyamatát.

7.4.1. Példa: Vegyünk fel interfészeket a naplózás elérésének segítésére.

+ Shop — fields ☐ methods + <i>aruk()</i> :Iterator + <i>vasarol(a:Aru, mennyiseg:long):void</i> + <i>getNev():String</i> + <i>getCim():String</i> + <i>getTulajdonos():String</i> + <i>nyit():void</i> + <i>zar():void</i> + <i>getLog():Log</i>	+ Log — fields ☐ methods + <i>getLogStream():InputStream</i> + <i>getVasarlasok():Iterator</i> + <i>getFeltoltesek():Iterator</i> + <i>getAruTorlesek():Iterator</i> + <i>getAruListaLekeresek():Iterator</i> + <i>getTejjesNaplozas():Iterator</i> + <i>getTejjesNaplozas.AsArray():LogBejegyzes[]</i>	+ LogBejegyzes — fields ☐ methods + <i>getDatum():Date</i> + <i>getLogInfo():String</i> + <i>isVasarlas():boolean</i> + <i>isFeltoltes():boolean</i> + <i>isAruTorles():boolean</i> + <i>isAruListaLekeres():boolean</i>
--	---	---

Így a bolt működése során bármikor lekérdezhetőek a naplózási információk, és az interfészeken keresztül könnyen feldolgozhatóak.

Lássuk az interfészek definícióját:

Log:

```

package bolt;

import java.io.InputStream;

```

```
import java.util.Iterator;

public interface Log {

    InputStream getLogStream();
    Iterator getVasarlasok();
    Iterator getFeltoltesek();
    Iterator getAruTorlesek();
    Iterator getAruListaLekereseke();
    Iterator getTeljesNaplozas();
    LogBejegyzes[] getTeljesNaplozasAsArray();

}
```

LogBejegyzés:

```
package bolt;

import java.util.Date;

public interface LogBejegyzes {

    Date getDatum();
    String getLogInfo();
    boolean isVasarlas();
    boolean isFeltoltes();
    boolean isAruTorles();
    boolean isAruListaLekeres();

}
```

Shop:

```
package bolt;
...
public interface Shop {
    ...
    Log getLog() throws ShopKivetel;
}
```

7.4.2 Példa: Interfészsel segítsük a napló létrehozását.

```
+ Logger
┌ fields
+ final FELTOLTES: int
+ final TORLES: int
+ final VASARLAS: int
+ final ARULISTALEKERES: int
└ methods
+ addVasarlas(info: String): void
+ addAruTorles(info: String): void
+ addAruFeltoltes(info: String): void
+ addAruListaLekerdese(info: String): void
+ closeLogging(): void
```

Definíciója:

```

package bolt;

public interface Logger {

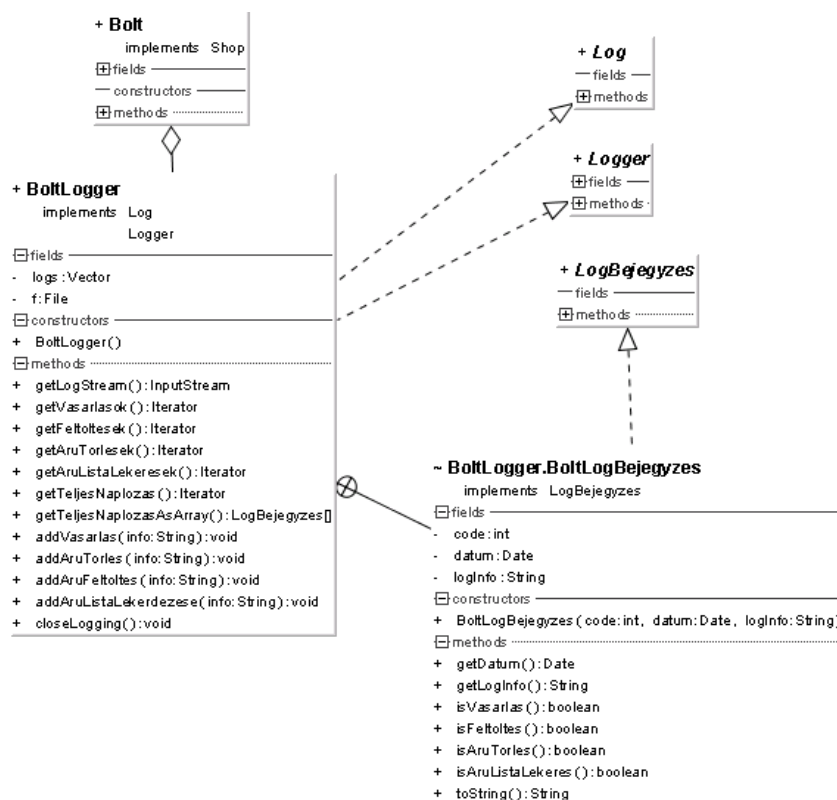
    static final int FELTOLTES = 1;
    static final int TORLES = 1;
    static final int VASARLAS = 1;
    static final int ARULISTALEKERES = 1;

    void addVasarlas(String info);
    void addAruTorles(String info);
    void addAruFeltoltes(String info);
    void addAruListaLekerdese(String info);
    void closeLogging();

}

```

7.4.3. Példa: Készítsük el a naplózási rész implementációját.



Egyazon osztály szolgálja ki a belső igényeket és a külsőket is. Nincs szükség a feladat megosztására, mert az felesleges idő-, tárigénnyel járna.
Lássuk az implementációt:

```

package bolt;
// imports
public class BoltLogger implements Log, Logger{

    private Vector logs;
    private File f;

```

```

public BoltLogger() {
    logs = new Vector();
}

class BoltLogBejegyzes implements LogBejegyzes{
    private int code;
    private Date datum;
    private String logInfo;

    public BoltLogBejegyzes(int code, Date datum, String logInfo) {
        this.code = code;
        this.datum = datum;
        this.logInfo = logInfo;
    }

    public Date getDatum() {
        return datum;
    }

    public String getLogInfo() {
        return logInfo;
    }

    public boolean isVasarlas() {
        return code == Logger.VASARLAS;
    }

    public boolean isFeltoltes() {
        return code == Logger.FELTOLTES;
    }

    public boolean isAruTorles() {
        return code == Logger.TORLES;
    }

    public boolean isAruListaLekeres() {
        return code == Logger.ARULISTALEKERES;
    }

    public String toString(){
        return logInfo+" dátuma:"+datum;
    }
}

public InputStream getLogStream() throws ShopKivetel{
    if(f==null) throw new ShopKivetel("A log meg nincs lezarva");
    try {
        return new FileInputStream(f);
    } catch (FileNotFoundException e) {
        throw new ShopKivetel("I/O hiba");
    }
}

public Iterator getVasarlasok() {
    Vector vasarlasok = new Vector();
    for (int i = 0; i < logs.size(); i++) {
        LogBejegyzes o = (LogBejegyzes) logs.elementAt(i);
        if(o.isVasarlas()) vasarlasok.add(o);
    }
}

```

```

        return vasarlasok.iterator();
    }

    public Iterator getFeltoltesek() {
        Vector vasarlasok = new Vector();
        for (int i = 0; i < logs.size(); i++) {
            LogBejegyzes o = (LogBejegyzes) logs.elementAt(i);
            if(o.isFeltoltes()) vasarlasok.add(o);
        }
        return vasarlasok.iterator();
    }

    public Iterator getAruTorlese() {
        Vector vasarlasok = new Vector();
        for (int i = 0; i < logs.size(); i++) {
            LogBejegyzes o = (LogBejegyzes) logs.elementAt(i);
            if(o.isAruTorles()) vasarlasok.add(o);
        }
        return vasarlasok.iterator();
    }

    public Iterator getAruListaLekerese() {
        Vector vasarlasok = new Vector();
        for (int i = 0; i < logs.size(); i++) {
            LogBejegyzes o = (LogBejegyzes) logs.elementAt(i);
            if(o.isAruListaLekeres()) vasarlasok.add(o);
        }
        return vasarlasok.iterator();
    }

    public Iterator getTeljesNaplozas() {
        return logs.iterator();
    }

    public LogBejegyzes[] getTeljesNaplozasAsArray() {
        return (LogBejegyzes[])logs.toArray( new LogBejegyzes[ logs.size()
] );
    }

    public void addVasarlas(String info) {
        logs.add( new BoltLogBejegyzes(Logger.VASARLAS, new Date(), info)
);
    }

    public void addAruTorles(String info) {
        logs.add( new BoltLogBejegyzes(Logger.TORLES, new Date(), info) );
    }

    public void addAruFeltoltes(String info) {
        logs.add( new BoltLogBejegyzes(Logger.FELTOLTES, new Date(), info)
);
    }

    public void addAruListaLekerdezese(String info) {
        logs.add( new BoltLogBejegyzes(Logger.ARULISTALEKERES, new Date(),
info) );
    }

    public void closeLogging() {
        StringBuffer sb = new StringBuffer();
        for (int i = 0; i < logs.size(); i++) {

```

```

        BoltLogBejegyzes boltLogBejegyzes = (BoltLogBejegyzes)
logs.elementAt(i);
        sb.append(boltLogBejegyzes+"");
    }
    try{
        f = new File("/path/log.txt");
        FileWriter w = new FileWriter(f);
        w.write(sb.toString());
        w.close();
    }
    catch(Exception e){ }
}
}

```

7.4.4. Példa: Építsük bele implementációnkat a Bolt osztályba!

```

package bolt;
...
public class Bolt implements Shop{

    private BoltLogger logger;
    ...
    public void nyit() {
        nyitva = true;
        logger = new BoltLogger();
    }
    public void zar() {
        nyitva = false;
        logger.closeLogging();
    }
    public Log getLog() throws ShopKivetel{
        return logger;
    }
    ...
    private boolean vanMegAdottAru(Class c){
        logger.addAruListaLekerdese("Árutípus készletezettségének
ellenőrzése:"+c);
        ...
    }
    public void feltoltElelmiszerrel(Long vonalKod, long mennyiseg) throws
BoltKivetel{
        logger.addAruTorles("Elelmiszerfeltoltes a"+vonalkod+" arubol
"+mennyiseg+" mennyiseggel");
        ...
    }
    public void feltoltUjElelmiszerrel(Elelmiszer e, long mennyiseg, long
ar){
        logger.addAruTorles("Új élelmiszer feltoltese:"+e+" "+mennyiseg+"
mennyiseggel");
        ...
    }
    public void torolElelmiszert(Long vonalKod) throws BoltKivetel{
        logger.addAruTorles("Elelmiszertorles:"+vonalkod);
        ...
    }
    public void vasarolElelmiszert(Long vonalKod, long mennyiseg) throws
BoltKivetel{

```

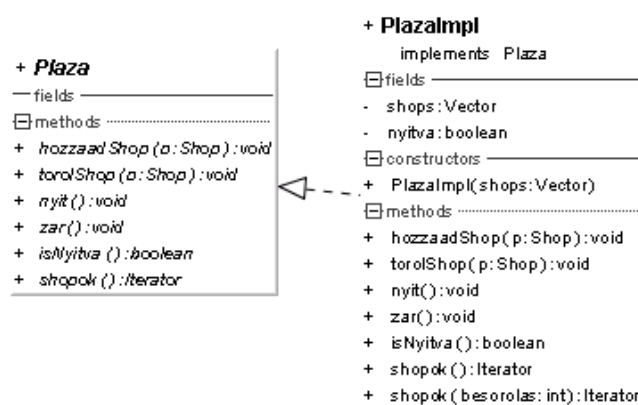


```

        logger.addVasarlas("Elelmiszervásárlás a"+vonalkod+" arubol
"+mennyiseg+" mennyiseggel");
        ...
    }
}

```

8. Példa: A személyek vásárlásának elősegítésére plázákat építenek, így rövidül a bevásárlási idő. Valósítsuk meg a plaza fogalmat Java környezetben! Az egyszerűség kedvéért feltételezhetjük, hogy a plaza csak boltokat tartalmaz, és azok egyszerre nyitnak. A boltokat funkciójuk szerint osztályozhatjuk: élelmiszerbolt, barkácsbolt, stb.



Implementáció:

Plaza:

```

package bolt;

import java.util.Iterator;

public interface Plaza{

    void hozzáadShop(Shop p);
    void torolShop(Shop p);
    void nyit();
    void zar();
    boolean isNyitva();
    Iterator shopok();

}

```

PlazaImpl:

```

package bolt;

import java.util.Iterator;
import java.util.Vector;

public class PlazImpl implements Plaza{

    private Vector shops;
    private boolean nyitva=false;

```

```

public PlazImpl(Vector shops) {
    this.shops = shops;
}

public void hozzaadShop(Shop p) {
    shops.add(p);
}

public void torolShop(Shop p) {
    shops.remove(p);
}

public void nyit() {
    nyitva = true;
    for (int i = 0; i < shops.size(); i++) {
        Shop shop = (Shop) shops.elementAt(i);
        shop.nyit();
    }
}

public void zar() {
    nyitva = false;
    for (int i = 0; i < shops.size(); i++) {
        Shop shop = (Shop) shops.elementAt(i);
        shop.zar();
    }
}

public boolean isNyitva() {
    return nyitva;
}

public Iterator shopok() {
    return shops.iterator();
}

public Iterator shopok(int besorolas) {
    Vector shopk = new Vector();
    for (int i = 0; i < shops.size(); i++) {
        Shop shop = (Shop) shops.elementAt(i);
        if( shop.getBesorolas()==besorolas ) shopk.add( shop );
    }
    return shopk.iterator();
}
}

```

Shop:

```

package bolt;

import java.util.Iterator;
import java.io.File;

public interface Shop {

    static final int ELELMISZER=1;
    static final int BARKACS=2;
    static final int RUHAZAT=3;

```

```

        static final int GYORSETTEREM=4;
        static final int KONYVESBOLT=5;
        ...
        int getBesorolas();
    }

    Bolt:
    ...
    public class Bolt implements Shop{
        ...
        private int besorolas;
        ...
        public Bolt(String nev, String tulajdonos, String cim, int besorolas) {
            this(nev, tulajdonos, cim, new Hashtable(), besorolas);
        }

        public Bolt(String nev, String tulajdonos, String cim, Hashtable
arupult, int besorolas) {
            this.nev = nev;
            this.tulajdonos = tulajdonos;
            this.cim = cim;
            this.arupult = arupult;
            this.besorolas = besorolas;
        }
        ...
        public int getBesorolas() {
            return besorolas;
        }
        ...
    }

```

Feladatok:

1. Módosítsuk a *Tej* osztályt, hogy meg tudja mondani egy *Tej* objektum, hogy mennyi ásványi anyagot és vitamint tartalmaz!
2. A *Tej* objektum sztring reprezentációjában is jelenjen meg az alkotóelem-információ!
3. Valósítsuk meg a *ZacskósTej* fogalmat!
4. Valósítsuk meg a *Kenyér*, *Kalács*, *Zsemle* fogalmat!
5. Valósítsuk meg a *Tejtermék* fogalmat!
6. Valósítsuk meg a *Pékáru* fogalmat!
7. A bolt kivétellel jelezzük a szabálytalan példányosításokat! (például: rossz zsírtartalom érték, vagy rossz úrtartalom érték, stb.)
8. A *Shop* zárását követően (és csak is akkor!) le lehessen kérdezni az aznap vásárolt áruk listáját.
9. Az aznap vásárolt áruk listáját a szabványos *Iterator* interfész egy implementációján keresztül adjuk át. Az iteráció elemei csak az áru nevét, kódját, eladott mennyiségét tartalmazzák!
10. A *Plaza* a nyitva lévő boltok listáját adja meg a *getNyitvaShopok()* metóduson keresztül

Egyéb feladatok:

1. Írj metódust, mely egy sztringről eldönti, hogy tartalmaz-e részsstringként valós számot!
2. Írj metódust, mely egy sztringről eldönti, hogy tartalmaz-e részsstringként nevet!
3. Írj metódust, mely egy sztringről eldönti, hogy tartalmaz-e részsstringként telefonszámot!
4. Írj metódust, mely egy sztringből kiemeli azon részsstringeket, melyek pontosan annyi magánhangzót tartalmaznak mint a teljes sztring.
5. Írj *encode()* metódust, mely egy sztringet titkosít a következő képpen:
Válasszon ki egy random számot 1 és az (angol) ABC elemszáma között.
Az eredménystring első karaktere e random szám ABC-ben kijelölt karaktere, és a bemeneti sztring minden karakterének ezen számmal eltol ABC karaktere.
Például: Input = „PELDA”
Random érték: 2
Eredménystring = „BNCJBY”
6. Írj *decode()* metódust, mely az előző metódus által titkosított sztringet visszaalakítja az eredetire.
Például: Input = „BNCJBY”
Random érték = B pozíciója: 2
Eredménystring = „PELDA”
7. Módosítsd az 1.5-ös feladatot, hogy ezt az eltolásos titkosítást *byte* tömbre végezze el!
8. Módosítsd ennek megfelelően az 1.6.-ban megírt metódust, hogy az 1.7.-es metódus által előállított sztringet vissza tudja fejteni.

9. Írjunk metódust, mely egy állományt olvas be, és kiemeli belőle azokat a szavakat, melyek nem kötőszavak (és, vagy, hogy, stb)!

10. Írjunk metódust, mely egy állományt olvas be, és kiemeli belőle azokat a sorokat, melyek nem kötőszavak (és, vagy, hogy, stb)!

11. Módosítsd az előző programot, hogy minden *Reader* objektumra működjön!

12. Módosítsd az előző programot, hogy minden *InputStream* objektumra működjön!

13. Legyen adott egy állomány, mely név-érték párosokat tartalmaz. Például:

PATH = /usr/lib/

Írj metódust, mely feldolgozza az ilyen szerkezetű állományt!

A következő metódusokkal rendelkezzen:

- public String getErtek(String nev);
- public String[] getNevek();
- public Iterator getNevek();
- public String getErtekek();
- public Iterator getErtekek();

14. Módosítsd az előző feladat metódusát, hogy többszörös név előfordulásokat is kezeljen, és előfordulásait előfordulásuk sorszámával lehessen lekérdezni:

- public String getErtek(String nev, int index);
- public int getElofordulasokSzama(String nev);
- public String[] getMindenErtek(String nev);
- public Iterator getMindenErtek(String nev);

Például

Bemenet:

PATH = /usr/lib

PATH = /usr/usrlib

Esetén a getErtek(„PATH”, 0); hívás a „/usr/lib” értékkel térjen vissza.

15. Módosítsd az előző programot, hogy az osztály ne tegyen különbséget kis és nagybetűk között

16. Módosítsd az előző programot, hogy megengedje az esetleges halmazértékűséget is!

Például:

USER = hallgato1, halgato2

Ez lényegében megegyezik a következő esettel:

USER = halgato1

USER = hallgato2

17. Írj Dátum típust, mely a következő metódusokkal rendelkezik:

- public Date(int ev, int ho, int nap, int ora, int perc, int masodperc);
- public Date(long ezredmasodperc); // 1970.01.01 0:00:00-tól számítva
- public int getEv();

- `public int getHo();`
- `public int getNap();`
- `public int getOra();`
- `public int getPerc();`
- `public int getMasodperc();`

18. Módosítsd az előző programot, hogy sztringből is ki tudja szedni az információt. A sztring formátuma: 00.00.00 00:00:00 (melyek rendre: ev.ho.nap ora:perc:mp)
Minden érték két karakter!

19. Módosítsd az előző programot, hogy sztring bemenet esetén több formátumot is megengedjen:

- `ev.ho.nap ora:perc:mp`
- `ev.ho.nap`
- `ora.perc.mp`
- `nap.ho.ev, ora:perc.mp`

20. Módosítsd az előző programot, hogy Dátum osztályod rendelkezzen a következő metódusokkal:

- `public void isKorabbi(Datum d);`
- `public void isKesobbj(Datum d);`
- `public Datum kulonbseg(Datum d);`
- `public Date osszegzes(Datum d);`

21. Módosítsd az előző programot, hogy kivétellel jelezze, hogy illegális formátumú inputot kap
(Minden konstruktora `IllegalFormat Exception`-t dobhat)

22. Módosítsd az előző programot, hogy Dátum osztályod rendelkezzen a következő metódusokkal:

- `public Date addNap(int nap);`
- `public Date addEv(int ev);`
- `public Date addHo(int ho);`
- `public Date addOra(int ora);`
- `public Date addPerc(int perc);`
- `public Date addMp(int mp);`

4. Zárthelyi dolgozatok példamegoldásai

A Debreceni Egyetemen az 1998/99-es tanév óta van lehetőség a „Programozás 2” tárgy gyakorlatát Java nyelven teljesíteni. A 2000/01-es tanévtől kezdve a nappali tagozaton „Programozás 2” gyakorlaton csak a Java nyelvet használjuk. Ezeken a gyakorlatokon számos zárthelyi dolgozatot (ZH-t) írtak már a diákok. Ezek feladatai közül oldunk itt meg néhányat.

Az első alkalmakkor a gyakorlatokon megíratott ZH-k feladatsorait az egyes gyakorlatvezetők egymástól függetlenül állították össze. A 2002/03-as tanévtől azonban központosított ZH-kat kellett a hallgatóknak teljesíteniük. Ez egy félévben 3 feladtsort jelent.

A központi ZH-kat megelőző időszakból mindössze két feladatot választottunk, míg a központi ZH-k valamennyi feladatsorával foglalkozunk. A központi feladatokból két feladatot hagytunk ki, melyek egyszerű hibajavítási feladatok és mint ilyenek nem túl jelentősek (számítógép mellett ülve könnyen megtalálhatók a hibák). Két feladatot apró módosítással adunk fel és oldunk meg, ezekenél a feladatoknál a módosítást jelöljük.

Megjegyezzük azt is, hogy korábbi dolgozatok feladatsorai [Prog2]-n megtalálhatók.

2000/01 I. félév

1. Feladat

Adott a következő osztály:

```
public abstract class Fuggveny {  
    public abstract double getErtek(double x);  
}
```

- a) Készítse el az adott `double[]` együtthatók együtthatókkal megadott `Polinom` függvényt ennek az osztálynak a pontosításaként. (azaz olyan `Polinom` osztályt, amelyet a `Fuggveny` osztályból származtatva kapunk és amely konstruktorának egy `double[]` együtthatók paramétere van, értékül pedig a `polinom x` helyen vett helyettesítési értékét adja meg).

Pl.: $0.4x^3 + x + 2$ polinom értékét a 4.5 helyen ezután így kapjuk:

```
new Polinom(new double[] {2, 1.0, 0.0, 0.4}).getErtek(4.5)
```

Tekintsük a következő interfészt:

```
interface Derivalhato {  
    Fuggveny getDerivaltoFuggveny();  
}
```

- b) Valósítsa meg az interfészt a `Polinom` osztályra
- c) Készítse el az adott `double a` együtthatós `Szinusz` és `Koszinusz` függvényeket ennek az osztálynak a pontosításaként. (azaz olyan `Szinusz` és `Koszinusz` osztályokat, melyeket a `Fuggveny` osztályból származtatva kapunk és amelyek konstruktorainak egy `double a` paramétere van, értékül pedig a `szinusz` ill. a `koszinusz` értékek `a`-szorosát szolgáltatják).
- d) Valósítsa meg az interfészt a `Szinusz` és `Koszinusz Fuggveny`-osztályokra

Megoldás

a) - b)

```
public class Polinom extends Fuggveny implements Derivalhato {  
  
    private double[] egyutthatok;  
  
    public Polinom(double[] egyutthatok) {
```

```

        this.egyutthatok = new double[egyutthatok.length];
        System.arraycopy(egyutthatok, 0,
                           this.egyutthatok, 0, egyutthatok.length);
    }

    public double getErtek(double x) {
        double rv = 0.0;
        for (int i = egyutthatok.length-1; i>=0; i--) {
            rv *= x;
            rv += egyutthatok[i];
        }
        return rv;
    }

    public Fuggveny getDerivaltFuggveny() {
        int egyuthatokSzama = egyutthatok.length - 1;
        if (egyuthatokSzama < 0) egyuthatokSzama = 0;
        double[] ujEgyutthatok = new double[egyuthatokSzama];
        for (int i = 1; i < egyutthatok.length; i++) {
            ujEgyutthatok[i-1] = i * egyutthatok[i];
        }
        return new Polinom(ujEgyutthatok);
    }
}

```

Megjegyzések:

- A konstruktorban másolatot készítünk a paraméterről, hogy annak utólagos módosítása erre a példányra ne lehessen hatással. Nem kötelező így csinálni, alkalmazástól függ.
- A függvényértéket a *Horner*-séma alapján számoljuk ki.
- Polinom deriváltja is polinom csak a megfelelő együtthatótömböt kell származtatni.

c) - d)

```

public class Szinusz extends Fuggveny implements Derivalhato {

    private double a;

    public Szinusz(double a) {
        this.a = a;
    }

    public double getErtek(double x) {
        return a*Math.sin(x);
    }

    public Fuggveny getDerivaltFuggveny() {
        return new Koszinusz(a);
    }
}

public class Koszinusz extends Fuggveny implements Derivalhato {

    private double a;

    public Koszinusz(double a) {
        this.a = a;
    }

    public double getErtek(double x) {
        return a*Math.cos(x);
    }
}

```



```

    public Fuggveny getDerivaltFuggveny() {
        return new Szinusz(-a);
    }
}

```

Megjegyzések:

- A kölcsönös hivatkozás miatt mindkét osztálynak rendelkezésre kell állnia fordításkor (de két állományban kell őket elhelyezni).
- $(a * \sin(x))' = a * \cos(x)$ ill. $(a * \cos(x))' = (-1) * a * \sin(x)$.

2. Feladat

Adott a következő osztály:

```

public class Tartaly {
    private double tartalom = 0;
    private double terfogat;

    public Tartaly(double terfogat) {
        this.terfogat = terfogat;
    }
}

```

Készítse el a következő módszereket:

```

public double getSzabadTerfogat()
public void beletolt(double mennyiseg) throws MegteltException
public void leereszt(double mennyiseg) throws
KifogyottException

```

A beletolt ill. leereszt módszerek jelezzék kivétellel, amennyiben már nem lehetséges az adott művelet végrehajtása. A kivételosztályok definícióját is adja meg!

Megoldás

Először a kivételosztályokat adjuk meg:

```

public class MegteltException extends Exception {
    public MegteltException() { super(); }
    public MegteltException(String message) { super(message); }
}

public class KifogyottException extends Exception {
    public KifogyottException() { super(); }
    public KifogyottException(String message) { super(message); }
}

```

A Tartaly implementálásakor már csak a megfelelő kivételek kiváltására kell odafigyelni.

```

public class Tartaly {
    private double tartalom = 0;
    private double terfogat;

    public Tartaly(double terfogat) {
        this.terfogat = terfogat;
    }

    public double getSzabadTerfogat() {
        return terfogat - tartalom;
    }

    public void beletolt(double mennyiseg) throws MegteltException {

```

```

        if (getSzabadTerfogat() < mennyiseg) {
            throw new MegteltException("A tartályba nem fér már ennyi.");
        }
        tartalom += mennyiseg;
    }

    public void leereszt(double mennyiseg) throws KifogyottException {
        if (tartalom < mennyiseg) {
            throw new KifogyottException("A tartályban nincs ennyi.");
        }
        tartalom -= mennyiseg;
    }
}

```

2002/03 I. félév

1. Feladatsor

1. Feladat

Adott két osztály. Az egyik törteket, a másik speciális törteket (egész számokat) reprezentál.

```

class Tort {
    protected int szamlalo, nevezo;

    public Tort(int sz, int n) {
        szamlalo = sz;
        nevezo = n;
    }

    public double ertek() {
        return (double) szamlalo / (double) nevezo;
    }

    public void szoroz(Tort t) {
        szamlalo *= t.szamlalo;
        nevezo *= t.nevezo;
    }
}

```

```

class Egesz extends Tort {
    public Egesz(int n) {
        super(n, 1);
    }

    public double ertek() {
        return szamlalo;
    }

    public void szoroz(int eg) {
        szamlalo *= eg;
    }
}

```

Adottak a következő változók:

```

Tort e = new Egesz(3);
Egesz f = new Egesz(5);
Tort t = new Tort(2,3);

```

Írja le, hogy hogyan szorozná meg 4-gyel az e, f, t változókkal hivatkozott számokat!

Megoldás

Az `e` változó lehetséges megszorzásai:

1. `e.szoroz(new Egesz(4));`
2. `e.szoroz(new Tort(4, 1));`
3. `((Egesz) e).szoroz(4);`

Az `e` referenciaváltozó önmagában vett típusa `Tort`, ezért csak a `Tort` osztály interfészébe is beleeső `Tort` paraméterű `szoroz` metódus használata lehetséges (1. és 2.). Mivel egy `Egesz` példány példánya a `Tort` osztálynak is, ezért az 1. megoldás is helyes.

Mivel az `e` által hivatkozott objektum dinamikus típusa `Egesz`, ezért explicit típuskonverzióval elérhetővé válik az `Egesz` osztály interfésze is és használható lesz az `int` paraméterű, névtülterhelt `szoroz` metódusváltozat (3.).

Megjegyzés:

- A témához kapcsolódik, hogy az 1. és 2. megoldásban a metódusok hívása késői kötéssel történik. Így hívás során az `Egesz` osztálybeli, felüldefiniált kód fut le. De ez most nem volt kérdés.

Az `f` változó lehetséges megszorzásai:

1. `f.szoroz(new Egesz(4));`
2. `f.szoroz(new Tort(4, 1));`
3. `f.szoroz(4);`

Az `f` referenciaváltozó típusa `Egesz`, ezért rendelkezik mind `Tort`, mind `int` paraméterű `szoroz` metódussal. Ezek bármelyike használható. Az 1. megoldás kihasználja, hogy minden `Egesz` egyben `Tort` is.

Az `t` változó lehetséges megszorzásai:

1. `t.szoroz(new Egesz(4));`
2. `t.szoroz(new Tort(4, 1));`

Az `t` referenciaváltozó típusa `Tort`, ezért csak `Tort` paraméterű `szoroz` metódussal rendelkezik. Az 1. megoldás kihasználja, hogy minden `Egesz` egyben `Tort` is.

Megjegyzés:

- Itt nem lehet típuskonverziót alkalmazni, mert a hivatkozott objektum dinamikus típusa is `Tort`, a konverziót nem lehetne futási időben végrehajtani.

2. Feladat

- a) Egészítse ki a kódot! Írjon olyan osztályt, amely egy egész típusú változó reprezentál. A változó számolja, hogy hányszor adtak értéket az adott változónak (hányszor változtatták meg az értékét).

Egészítse ki az osztályt úgy, hogy a változó képes legyen visszaadni azt, hogy hány módosítás történt az értékén.

```
class Variable{
    protected int value;

    public Variable(int initial) {
        value = initial;
    }

    public int getValue() {
        ...
    }
}
```

```
public void setValue(int newValue) {  
    ...  
}  
}
```

- b) Írjon statikus metódust, amely megadja, hogy a `Variable` osztálynak hány jelenleg élő példánya van!

Megoldás

a) - b)

```
class Variable {  
  
    protected int value;  
    protected int changeCount;  
  
    private static int instanceCount = 0;  
  
    public Variable(int initial) {  
        value = initial;  
        changeCount = 0;  
    }  
  
    public int getValue() {  
        return value;  
    }  
  
    public void setValue(int newValue) {  
        value = newValue;  
        changeCount++;  
    }  
  
    public int getChangeCount() {  
        return changeCount;  
    }  
  
    protected void finalize() {  
        instanceCount--;  
    }  
  
    public static int getInstanceCount() {  
        return instanceCount;  
    }  
  
}
```

Az a) rész megvalósításához felvettük az osztályba a `changeCount` attribútumot, amiben a `setValue(...)` hívásokat számláljuk. Az attribútumhoz megadtunk a lekérdező metódusát `getChangeCount()`.

A b) rész példányszámlálását az `instanceCount` statikus attribútummal valósítjuk meg, mely értékét a `getInstanceCount()` metódus adja meg. A számláló értékét növeljük minden példányosításkor és csökkentjük a felüldefiniált `finalize()` metódusban, mielőtt az objektumot a szeméthyűjtő rendszer (garbage collector) törli a memóriából.

Megjegyzések:

- Az a) résznél félreérthető, hogy pontosan mikor kell növelni a `changeCount` számlálót. Minden `setValue()` híváskor vagy csak akkor, ha tényleg megváltozik a változó értéke. Ezért a következő változat is elfogadható:

```
public void setValue(int newValue) {
```

```

        if (value != newValue) {
            value = newValue;
            changeCount++;
        }
    }
}

```

- A b) részben a `finalize()` metódustól esetleg eltekinthetünk.

2. Feladatsor

1. Feladat

Írjon osztályt, amely a `Muvelet` interfész segítségével minden ügyfél számlaegyenlegét növeli 5%-kal, ha az egyenleg pozitív és csökkenti 10%-kal, ha az egyenleg negatív. A `kamatotTokesit` metódusnak helyesen kell működnie!

```

interface Muvelet {
    public Object feldolgoz(Object o);
}

```

```

public class BankSzamlak {
    protected String[] ugyfel;
    protected Double[] egyenleg;

    ...

    public void kamatotTokesit(Muvelet m) {
        for (int j = 0; j < ugyfel.length; j++)
            egyenleg[j] = (Double) m.feldolgoz(egyenleg[j]);
    }

    ...
}

```

Megoldás

Több megoldást adunk.

1. megoldás

Implementáljuk a `Muvelet` interfészt egy osztállyal, valamint az osztályban adjunk meg egy `BankSzamlak` paraméterű metódust. Ez a metódus a paraméterül kapott számlák `kamatotTokesit` metódusát helyesen paraméterezve meghívja.

```

public class KamatMuvelet implements Muvelet {
    public Object feldolgoz(Object o) {
        double egyenleg = ((Double) o).doubleValue();
        return new Double(egyenleg * (egyenleg > 0 ? 1.05 : 1.10));
    }

    public void kamatotTokesit(BankSzamlak b) {
        b.kamatotTokesit(this);
    }
}

```

2. megoldás

Megadunk egy osztályt, amelyben lesz egy `BankSzamlak` paraméterű metódus. Ez a metódus a paraméterül kapott számlák `kamatotTokesit` metódusát helyesen paraméterezve meghívja. A paraméter itt egy névtelen osztály példánya lesz:

```

public class KamatUtil {

```

```

    public void kamatotTokesit(BankSzamlak b) {
        b.kamatotTokesit(new Muvelet(){
            public Object feldolgoz(Object o) {
                double egyenleg = ((Double) o).doubleValue();
                return new Double(egyenleg * (egyenleg > 0 ? 1.05 : 1.10));
            }
        });
    }
}

```

3. megoldás

Implementáljuk a `Muvelet` interfészt egy osztállyal. Egy másik osztályban pedig adjunk meg egy `BankSzamlak` paraméterű metódust. Ez a metódus a paraméterül kapott számlák `kamatotTokesit` metódusát helyesen paraméterezve meghívja. A paraméter itt az első osztály egy példánya lesz:

```

public class KamatMuvelet implements Muvelet {
    public Object feldolgoz(Object o) {
        double egyenleg = ((Double) o).doubleValue();
        return new Double(egyenleg * (egyenleg > 0 ? 1.05 : 1.10));
    }
}

public class KamatUtil {
    public void kamatotTokesit(BankSzamlak b) {
        b.kamatotTokesit(new KamatMuvelet());
    }
}

```

2. Feladat (módosított változat)

Írjon rendezett láncolt lista osztályt, amelyben összehasonlítható elemek vannak. Valósítsa meg a beszúrás és a keresés műveleteket, továbbá írjon műveletet, amely visszaadja a lista felsorolását. Dobja el a szükséges kivételeket:

- a listában nem lehetnek egyező elemek,
- a keresett elem nincs a listában.

```

interface Hasonlithato {
    public int hasonlit(Hasonlithato h);
}

```

```

interface Felsorolas {
    public Hasonlithato kovetkezo();
    public boolean vanMeg();
}

```

- a) Készítsen `Hallgato` nevű osztályt, amely a hallgatók nevét, évfolyamát, átlagát tárolja, úgy, hogy ezen osztály példányait a listában lehessen tárolni. A hallgatók név és évfolyam szerint hasonlítandók össze.

Továbbá adjon meg a `Hallgatok` osztályban egy `feltolt` nevű statikus metódust, amely paraméterei:

```

LancoltLista lista
String[] nevek
int[] evfolyamok
double[] atlagok

```

Az ezekben lévő adatokat – mint hallgatókat – a rendezett listában eltárolja. Kezelje a szükséges kivételeket!

Megoldás

a)

A megoldás láncolt listájában csak a feladatban kért metódusokat adjuk meg (nem lesz például törlés). A `Hasonlithato` interfész `hasonlit(...)` metódusa a következőképpen működjön:

$$a.\text{hasonlit}(b) \quad \begin{cases} < 0 & a < b \\ = 0 & \Leftrightarrow a = b \\ > 0 & a > b \end{cases}$$

Kivételosztályok:

```
public class DuplikaltElemException extends Exception {
    public DuplikaltElemException() {}
    public DuplikaltElemException(String message) { super(message); }
}

public class NincsIlyenElemException extends Exception {
    public NincsIlyenElemException() {}
    public NincsIlyenElemException(String message) { super(message); }
}
```

A láncolt lista:

```
public class LancoltLista {

    private ListaElem fej = null;

    private static class ListaElem {
        Hasonlithato elem;
        ListaElem kov;

        public ListaElem(Hasonlithato elem, ListaElem kov) {
            this.elem = elem;
            this.kov = kov;
        }
    }

    public void beszur(Hasonlithato h) throws DuplikaltElemException {
        if (h == null) throw new NullPointerException();
        if (fej == null || h.hasonlit(fej.elem) < 0) {
            // Lista elejére kell beszúrni.
            fej = new ListaElem(h, fej);
        } else {
            ListaElem megalozo;
            for (megalozo = fej;
                megalozo.kov != null
                && h.hasonlit(megalozo.kov.elem) >= 0;
                megalozo = megalozo.kov)
                ;
            if (h.hasonlit(megalozo.elem) == 0) {
                throw new DuplikaltElemException();
            }
            megalozo.kov = new ListaElem(h, megalozo.kov);
        }
    }

    public int keres(Hasonlithato h) throws NincsIlyenElemException {
```

```

        int poz = 0;
        ListaElem e;
        for (e = fej, poz = 0;
             e != null && h.hasonlit(e.elem) > 0;
             e = e.kov, poz++)
            ;
        if (e != null && h.hasonlit(e.elem) == 0)
            return poz;
        else
            throw new NincsIlyenElemException();
    }

    private class ListaFelsorolas implements Felsorolas {
        ListaElem kovetkezoElem = fej;

        public Hasonlithato kovetkezo() {
            Hasonlithato rv = kovetkezoElem.elem;
            kovetkezoElem = kovetkezoElem.kov;
            return rv;
        }

        public boolean vanMeg() {
            return kovetkezoElem != null;
        }
    }

    public Felsorolas felsorolas() {
        return new ListaFelsorolas();
    }
}

```

A `LancoltLista` osztályban találhatók a `ListaElem` beágyazott és `ListaFelsorolas` belső osztályok is. A `ListaElem` megadható lett volna független osztályként vagy nem statikusan, belső osztályként. A `ListaFelsorolas` osztályt pedig névtelen osztállyal is helyettesíthetnénk. (Hasonlóan a 2003/04-es év 2. feladatsorának 3. feladatához.)

A listaműveletek algoritmusainak elemzését az olvasóra hagyjuk, feltételezve előző ismereteit.

b)

A `Hallgato` osztálynak implementálnia kell a `Hasonlithato` interfészt, hogy a példányok a listában elhelyezhetők legyenek.

```

public class Hallgato implements Hasonlithato {
    private String nev;
    private int evfolyam;
    private double atlag;

    public Hallgato(String nev, int evfolyam, double atlag) {
        this.nev = nev;
        this.evfolyam = evfolyam;
        this.atlag = atlag;
    }

    public int hasonlit(Hasonlithato h) {
        Hallgato masik = (Hallgato) h;
        int cmp = nev.compareTo(masik.nev);
        if (cmp != 0)
            return cmp;
        else if (evfolyam < masik.evfolyam)
            return -1;
        else if (evfolyam > masik.evfolyam)

```



```

        return 1;
    else
        return 0;
}

public static void feltolt(LancoltLista lista,
                          String[] nevek,
                          int[] evfolyamok,
                          double[] atlagok)
{
    for (int i = 0; i < nevek.length; i++) {
        try {
            lista.beszur(new Hallgato(nevek[i], evfolyamok[i],
                                      atlagok[i]));
        } catch (DuplikaltElemException ex) {
            System.err.println(nevek[i] + " nevű " + evfolyamok[i]
                               + " évf. hallgató ismételt szerepel.");
        }
    }
}

public String toString() {
    return "Hallgató: " + nev + ", " + evfolyam + ", " + atlag;
}
}

```

Nem kért ugyan a feladat sem ellenőrző osztályt sem `toString` metódus megadását a `Hallgato` osztályban, de ha a kódot ellenőrizni szeretnénk, akkor ezekre szükség van.

Ellenőrzés

A következőkben, annak ellenére, hogy nem volt feladat, megadjuk a fenti kódokat tesztelő programot és kimenetét.

```

public class ListaTeszt {

    private static void prompt(int i) {
        System.out.println("\n" + i + ". teszt\n");
    }

    public static void main(String[] args) throws Exception {
        LancoltLista l1 = new LancoltLista();

        /*
         * 1. teszt
         * - l1 lista feltöltés elemekkel.
         */
        prompt(1);
        Hallgato.feltolt(l1,
                        new String[]{"Cecil", "Ferenc", "Anikó", "Dezső", "Anikó"},
                        new int[]{3, 2, 3, 1, 2},
                        new double[]{1.0, 2.0, 3.0, 4.0, 5.0}
        );

        /*
         * 2. teszt
         * - l1 lista felsorolása
         * - l1 listában tartalmazott elemek keresése.
         * - l2 lista részleges feltöltése.
         */
        prompt(2);
        Hallgato h;
    }
}

```

```

LancoltLista l2 = new LancoltLista();
int i = 0;
for (Felsorolas f = l1.felsorolas(); f.vanMeg(); i++) {
    h = (Hallgato) f.kovetkezo();
    System.out.println(l1.keres(h) + ": " + h);
    if (i % 2 == 0) {
        l2.beszur(h);
    }
}

/*
 * 3. teszt
 * - l2 listában tartalmazott és nem tartalmazott elemek keresése
 */
prompt(3);
for (Felsorolas f = l1.felsorolas(); f.vanMeg(); i++) {
    h = (Hallgato) f.kovetkezo();
    try {
        System.out.println("Van: " + l2.keres(h) + ": " + h);
    } catch (NincsIlyenElemException e) {
        System.out.println("Nincs:  " + h);
    }
}

/*
 * 4. teszt
 * - l2 listába tartalmazott és nem tartalmazott elemek beszúrása
 */
prompt(4);
for (Felsorolas f = l1.felsorolas(); f.vanMeg(); i++) {
    h = (Hallgato) f.kovetkezo();
    try {
        l2.beszur(h);
        System.out.println("Beszúrt:  " + l2.keres(h) + ": " + h);
    } catch (DuplikaltElemException e) {
        System.out.println("Már volt: " + l2.keres(h) + ": " + h);
    }
}
}
}

```

Kimenet

1. teszt

2. teszt

```

0: Hallgató: Anikó, 2, 5.0
1: Hallgató: Anikó, 3, 3.0
2: Hallgató: Cecil, 3, 1.0
3: Hallgató: Dezső, 1, 4.0
4: Hallgató: Ferenc, 2, 2.0

```

3. teszt

```

Van: 0: Hallgató: Anikó, 2, 5.0
Nincs: Hallgató: Anikó, 3, 3.0
Van: 1: Hallgató: Cecil, 3, 1.0
Nincs: Hallgató: Dezső, 1, 4.0
Van: 2: Hallgató: Ferenc, 2, 2.0

```

4. teszt

Már volt: 0: Hallgató: Anikó, 2, 5.0
Beszúrt: 1: Hallgató: Anikó, 3, 3.0
Már volt: 2: Hallgató: Cecil, 3, 1.0
Beszúrt: 3: Hallgató: Dezső, 1, 4.0
Már volt: 4: Hallgató: Ferenc, 2, 2.0

3. Feladat

Készítsen olyan osztályt, amely konstruktorában megkap egy szóközzel elválasztott különálló betűkből (ismeretlenek), természetes számokból és műveleti jelekből álló sztringet (+, -, *, /). Ez a sztring egy szabályos postfix kifejezést reprezentál, amelyben a műveleti jelek kétoperandusú operátorok. A konstruktor építse fel a kifejezésfát, majd határozza meg azt a legbonyolultabb részkifejezést, amely nem tartalmaz ismeretleneket (ha több ilyen is van, akkor a legbaloldalibbat kell meghatározni). A kifejezés bonyolultságát a kifejezéshez tartozó kifejezésfa magassága adja meg.

1. megoldás

A megoldás két osztályból áll, a fát reprezentáló `KifejezesFa` osztályból és az azt felépítő, kezelő `Postfix` osztályból.

```
public class KifejezesFa {
    public static final int T_OP = 0;
    public static final int T_VALTOZO = 1;
    public static final int T_SZAM = 2;

    /* Ez határozza meg, hogy az alábbi mezők mely értéke érvényes. */
    private int tipus;

    private String op;
    private String valtozo;
    private int ertek;

    private KifejezesFa bal;
    private KifejezesFa jobb;

    private int magassag;

    public static int tipusa(String tartalom) {
        try {
            Integer.parseInt(tartalom);
            return T_SZAM;
        } catch (NumberFormatException ex) {
            return Character.isLetter(tartalom.charAt(0))
                ? T_VALTOZO
                : T_OP;
        }
    }

    public KifejezesFa(int tipus, String tartalom,
                       KifejezesFa bal, KifejezesFa jobb) {
        this.tipus = tipus;
        switch (tipus) {
            case T_SZAM:
                ertek = Integer.parseInt(tartalom);
                magassag = 1;
        }
    }
}
```

```

        break;
    case T_VALTOZO:
        valtozo = tartalom;
        magassag = 1;
        break;
    case T_OP:
        op = tartalom;
        magassag = Math.max(bal.magassag, jobb.magassag) + 1;
        this.bal = bal;
        this.jobb = jobb;
        break;
    default:
        throw new IllegalArgumentException("Nem létező típus.");
    }
}

public KifejezesFa(int tipus, String tartalom) {
    this(tipus, tartalom, null, null);
}

public KifejezesFa(String tartalom) {
    this(tipusa(tartalom), tartalom);
}

public int getTipus() {
    return tipus;
}

public String getOp() {
    return op;
}

public String getValtozo() {
    return valtozo;
}

public int getErtek() {
    return ertekek;
}

public KifejezesFa getBal() {
    return bal;
}

public KifejezesFa getJobb() {
    return jobb;
}

public int getMagassag() {
    return magassag;
}
}

```

A `KifejezesFa` osztály attribútumai csak lekérdezhetők, egyébként privát láthatóságúak. Ezek láthatósága lehetne megengedőbb, így kevesebb kódot kellene írni (pl. amikor dolgozatban papírra kell írni). Minden kifejezésfa egy attribútumban (`tipus`) tárolja a csomópont típusát és ennek megfelelően 3 másik attribútum (`op`, `valtozo`, `ertekek`) közül mindig csak egy tartalmaz értékes adatot. OO környezetben egy olyan fa esetén, ahol a csomópontok típusa eltérő lehet használhatnánk öröklődést is a típusok megadására. Pl.:

```
public abstract class Kifejezes ...
```

```

public class ÖsszetettKifejezes extends Kifejezes ...
public class VáltozóKifejezes extends Kifejezes ...
public class KonstansKifejezes extends Kifejezes ...

```

Így nem lenne szükség típus attribútumra és az egyes attribútumok (op, valtozo, ertek) a megfelelő alosztályokba kerülnének csak bele. Mivel azonban a lehetséges típusok száma igen kevés, 3 db mindössze, ezért használhatjuk a hagyományos megközelítést is. (Ez a megoldás 2. megoldásként szerepel, további magyarázatok nélkül).

Minden fa tárolja a magasságát is a `magassag final` attribútumban. Ez megtehető, hiszen a magasság a kifejezésfában a létrehozás után már nem változik meg, nincs szükség annak dinamikus kiszámítására. A `magassag` ún. „blank final”, azaz kezdőérték nélküli konstans, melynek a konstruktorokban (vagy inicializáló blokkokban) pontosan egyszer értéket kell adni.

A fenti osztály példányaiból épülhet fel egy kifejezésfa. A fát felépítő és kezelő osztály a `Postfix` osztály nevet kapta:

```

import java.util.Stack;
import java.util.StringTokenizer;

public class Postfix {
    private KifejezesFa fa;
    private KifejezesFa reszFa;
    private String bonyolultReszKifejezes;

    public Postfix(String postfixKifejezes) {
        fa = postfixbolFa(postfixKifejezes);
        reszFa = legbonyolultabbValtozoMentesReszFa(fa);
        bonyolultReszKifejezes = toString(reszFa);
    }

    public KifejezesFa getFa() {
        return fa;
    }

    public KifejezesFa getReszFa() {
        return reszFa;
    }

    public String getBonyolultReszKifejezes() {
        return bonyolultReszKifejezes;
    }

    public static KifejezesFa postfixbolFa(String postfixKifejezes) {
        StringTokenizer st = new StringTokenizer(postfixKifejezes);
        Stack s = new Stack();
        int tipus;
        KifejezesFa bal, jobb;
        while (st.hasMoreTokens()) {
            String token = st.nextToken();
            tipus = KifejezesFa.tipusa(token);
            if (tipus == KifejezesFa.T_OP) {
                jobb = (KifejezesFa) s.pop();
                bal = (KifejezesFa) s.pop();
                s.push(new KifejezesFa(tipus, token, bal, jobb));
            } else {
                s.push(new KifejezesFa(tipus, token));
            }
        }
        return (KifejezesFa) s.pop();
    }
}

```

```

public static KifejezesFa legbonyolultabbJoReszFa(KifejezesFa fa) {
    KifejezesFa rv = null;
    switch (fa.getTipus()) {
        case KifejezesFa.T_SZAM:
            rv = fa;
            break;
        case KifejezesFa.T_VALTOZO:
            rv = null;
            break;
        case KifejezesFa.T_OP:
            KifejezesFa bResz = legbonyolultabbJoReszFa(fa.getBal());
            KifejezesFa jResz = legbonyolultabbJoReszFa(fa.getJobb());
            if (bResz == fa.getBal() && jResz == fa.getJobb()) {
                rv = fa;
            } else {
                if (magassag(bResz) >= magassag(jResz)) {
                    rv = bResz;
                } else {
                    rv = jResz;
                }
            }
            break;
    }
    return rv;
}

public static int magassag(KifejezesFa fa) {
    return fa == null ? 0 : fa.getMagassag();
}

public static String toString(KifejezesFa fa) {
    // postfix alakot allit elo
    if (fa == null) return "";
    if (fa.getTipus() == KifejezesFa.T_OP) {
        return toString(fa.getBal()) + " "
            + toString(fa.getJobb()) + " "
            + fa.getOp();
    } else {
        return fa.getTipus() == KifejezesFa.T_VALTOZO
            ? fa.getValtozo()
            : String.valueOf(fa.getErtek());
    }
}
}

```

A `PostFix` osztály konstruktora 3 feladatot hajt végre:

1. A megadott postfix kifejezésből felépíti a fát. Ez az algoritmus egy különálló nyilvános statikus metódusba került, mert önálló részfeladatként is értelmesen végrehajtható. A felépített fát tároljuk, ezután az lekérdezhető.
Az algoritmus az ilyenkor szokásos veremmel történő feldolgozás, amit itt nem részletezünk. A megoldás során használjuk az `API StringTokenizer` osztályát a szóközők menti felbontásra, valamint a `Stack` osztályt egyszerű veremként. (Nem kell a feladat megoldásához saját veremosztályt írni!) Ha valakinek idegen a `Stack`, használhat `Vector`-t vagy egy elegendően nagy tömböt is.
2. A felépített kifejezésfában meg kell határozni azt a legbonyolultabb részkifejezést, amely nem tartalmaz ismeretleneket. Ez egy részfa esetén attól függ, hogy milyen típusú csomópont a részfa gyökere:
 - maga a részfa, ha a gyökér egy érték (a gyökér a teljes részfa és ez jó is)

- üres fa (`null`), ha a gyökér egy változó (a gyökér a teljes részfa, de az nem jó)
- összetett kifejezésnél, ha a gyökérben egy műveleti jel van, akkor kell igazán dolgozni.

Az utolsó pontban, műveleti jelnél meghatározzuk a bal és jobboldali részfákban a legjobb kifejezést (rekurzió) és ha,

- ezek maguk a teljes bal- ill. teljes jobboldali részfák, akkor a teljes összetett kifejezés a legbonyolultabb
- egyébként a magasabbat kell választani, egyezés esetén pedig a baloldalit.

3. Mivel a feladat részkifejezést kért és nem részfát, ezért a megtalált részfát visszaalakítjuk postfix kifejezéssé. Ez egy egyszerű rekurzív algoritmussal megtehető

A feladat a 3. lépés nélkül is elfogadható.

2. megoldás

A megoldás az előzőtől a csomópontok típusának reprezentációjában tér el. Öröklődést használ és nem egy típus attribútumot, ezért további magyarázatok nélkül adjuk meg.

Kifejezes:

```
public abstract class Kifejezes {
    public static final int T_OP = 0;
    public static final int T_VALTOZO = 1;
    public static final int T_SZAM = 2;

    public abstract int getMagassag();

    public static int tipusa(String tartalom) {
        try {
            Integer.parseInt(tartalom);
            return T_SZAM;
        } catch (NumberFormatException ex) {
            return Character.isLetter(tartalom.charAt(0))
                ? T_VALTOZO
                : T_OP;
        }
    }

    public static Kifejezes buildKifejezes(int tipus, String tartalom,
        Kifejezes bal, Kifejezes jobb) {
        switch (tipus) {
            case T_SZAM:
                return new KonstansKifejezes(Integer.parseInt(tartalom));
            case T_VALTOZO:
                return new ValtozoKifejezes(tartalom);
            case T_OP:
                return new OsszetettKifejezes(tartalom, bal, jobb);
            default:
                throw new IllegalArgumentException("Nem létező típus.");
        }
    }

    public static Kifejezes buildKifejezes(int tipus, String tartalom) {
        return buildKifejezes(tipus, tartalom, null, null);
    }
}
```

KonstansKifejezes:

```

public class KonstansKifejezes extends Kifejezes {
    private int ertek;

    public KonstansKifejezes(int ertek) {
        this.ertek = ertek;
    }

    public int getErtek() {
        return ertek;
    }

    public int getMagassag() {
        return 1;
    }
}

```

ValtozoKifejezes:

```

public class ValtozoKifejezes extends Kifejezes {
    private String valtozo;

    public ValtozoKifejezes(String valtozo) {
        this.valtozo = valtozo;
    }

    public String getValtozo() {
        return valtozo;
    }

    public int getMagassag() {
        return 1;
    }
}

```

OsszetettKifejezes:

```

public class OsszetettKifejezes extends Kifejezes {

    private String op;

    private Kifejezes bal;
    private Kifejezes jobb;

    private final int magassag;

    public OsszetettKifejezes(String tartalom,
                               Kifejezes bal, Kifejezes jobb) {
        op = tartalom;
        magassag = Math.max(bal.getMagassag(), jobb.getMagassag())
            + 1;
        this.bal = bal;
        this.jobb = jobb;
    }

    public String getOp() {
        return op;
    }

    public Kifejezes getBal() {
        return bal;
    }

    public Kifejezes getJobb() {

```



```

        return jobb;
    }

    public int getMagassag() {
        return magassag;
    }
}

```

PostFix:

```

import java.util.Stack;
import java.util.StringTokenizer;

public class Postfix {
    private Kifejezes fa;
    private Kifejezes reszFa;
    private String bonyolultReszKifejezes;

    public Postfix(String postfixKifejezes) {
        fa = postfixbolFa(postfixKifejezes);
        reszFa = legbonyolultabbJoReszFa(fa);
        bonyolultReszKifejezes = toString(reszFa);
    }

    public Kifejezes getFa() {
        return fa;
    }

    public Kifejezes getReszFa() {
        return reszFa;
    }

    public String getBonyolultReszKifejezes() {
        return bonyolultReszKifejezes;
    }

    public static Kifejezes postfixbolFa(String postfixKifejezes) {
        StringTokenizer st = new StringTokenizer(postfixKifejezes);
        Stack s = new Stack();
        int tipus;
        Kifejezes bal, jobb;
        while (st.hasMoreTokens()) {
            String token = st.nextToken();
            tipus = Kifejezes.tipusa(token);
            if (tipus == Kifejezes.T_OP) {
                jobb = (Kifejezes) s.pop();
                bal = (Kifejezes) s.pop();
                s.push(Kifejezes.buildKifejezes(tipus, token, bal, jobb));
            } else {
                s.push(Kifejezes.buildKifejezes(tipus, token));
            }
        }
        return (Kifejezes) s.pop();
    }

    public static Kifejezes legbonyolultabbJoReszFa(Kifejezes fa) {
        Kifejezes rv = null;
        if (fa instanceof KonstansKifejezes) {
            rv = fa;
        } else if (fa instanceof ValtozoKifejezes) {
            // rv = null;
        } else if (fa instanceof OsszetettKifejezes) {

```

```

        OsszetettKifejezes ofa = (OsszetettKifejezes) fa;
        Kifejezes bResz = legbonyolultabbJoReszFa(ofa.getBal());
        Kifejezes jResz = legbonyolultabbJoReszFa(ofa.getJobb());
        if (bResz == ofa.getBal() && jResz == ofa.getJobb()) {
            rv = fa;
        } else {
            if (magassag(bResz) >= magassag(jResz)) {
                rv = bResz;
            } else {
                rv = jResz;
            }
        }
    } else {
        throw new IllegalArgumentException("Ismeretlen kifejezéstípus");
    }
    return rv;
}

public static int magassag(Kifejezes fa) {
    return fa == null ? 0 : fa.getMagassag();
}

public static String toString(Kifejezes fa) {
    // postfix alakot allit elo
    if (fa == null) return "";
    if (fa instanceof OsszetettKifejezes) {
        OsszetettKifejezes ofa = (OsszetettKifejezes) fa;
        return toString(ofa.getBal()) + " "
            + toString(ofa.getJobb()) + " "
            + ofa.getOp();
    } else {
        return fa instanceof ValtozoKifejezes
            ? ((ValtozoKifejezes) fa).getValtozo()
            : String.valueOf(((KonstansKifejezes) fa).getErtek());
    }
}
}

```

3. Feladatsor

1. Feladat

Írjon PROGRAMOT, amely argumentumaiban megkapja két létező állomány nevét. Az első állományban minden sorban egy oktató neve szerepel, úgy hogy egy oktató csak egyszer szerepel az állományban (tehát az oktatók nevei különbözőek). A második állomány sorai a következők:

Kováts Endre#Nagy Elemér

Minden sorban pontosan két név szerepel '#'-kal elválasztva. Az első név egy hallgató neve, a második pedig az oktatóé (az oktató nevét az első állomány tartalmazza). A program írja ki a képernyőre azon oktatók neveit, akik a legtöbb hallgatóval rendelkeznek, és azon oktatók neveit, akinek egyáltalán nincsenek hallgatóik

1. megoldás

Ebben a megoldásban a tanárok és a hallgatók kapcsolatát reprezentáló adatszerkezetet létrehozunk. Igaz ezt a szerkezetet nem használjuk ki csak két egyszerű kérdés megválaszolására, mégis a megoldás példaértékű, ha a feladat általánosabb. (A 2. megoldás konkrétan a létszámhoz kapcsolódó problémákra koncentrálnak.)

```

import java.io.*;
import java.util.*;

public class TanarHallgato {
    private static Hashtable tanarokHallgatoi;

    public static void main(String[] args) {
        tanarokHallgatoi = new Hashtable();
        FileReader in = null;
        try {
            readTanarok(new BufferedReader(in = new FileReader(args[0])));
            in = new FileReader(args[1]);
            readHallgatok(new BufferedReader(in));
        } catch (IOException e) {
            System.err.println("IO Hiba."
                + " Ellenőrizze a megadott állományokat!");
        } finally {
            try { in.close(); } catch (Exception e) {}
        }
        int maxLetszam = getMaxLetszam();
        System.out.println("Legtöbb hallgatóval (" + maxLetszam
            + ") rendelkező tanárok:");
        System.out.println("~~~~~");
        printTanarByHallgatoLetszam(maxLetszam, System.out);
        System.out.println();
        System.out.println("Hallgató nélküli tanárok:");
        System.out.println("~~~~~");
        printTanarByHallgatoLetszam(0, System.out);
    }

    private static void readTanarok(BufferedReader in) throws IOException {
        String s;
        while ((s = in.readLine()) != null) {
            tanarokHallgatoi.put(s, new HashSet());
        }
    }

    private static void readHallgatok(BufferedReader in)
        throws IOException {
        String s;
        String tanar;
        String hallgato;
        int hashmark;
        HashSet hallgatok;
        while ((s = in.readLine()) != null) {
            hashmark = s.indexOf("#");
            hallgato = s.substring(0, hashmark);
            tanar = s.substring(hashmark + 1);
            hallgatok = (HashSet) tanarokHallgatoi.get(tanar);
            // hallgatok nem lehet null, ha az állományok helyesek
            hallgatok.add(hallgato);
        }
    }

    private static int getMaxLetszam() {
        int max = 0;
        HashSet hallgatok;
        for (Iterator i = tanarokHallgatoi.values().iterator();
            i.hasNext();) {
            hallgatok = (HashSet) i.next();
        }
    }
}

```

```

        if (hallgatok.size() > max) {
            max = hallgatok.size();
        }
    }
    return max;
}

private static void printTanarByHallgatoLetszam(int letszam,
                                                PrintStream out) {
    String tanar;
    HashSet hallgatok;
    Map.Entry tanarEsHallgatok;
    for (Iterator i = tanarokHallgatoEntrySet().iterator();
         i.hasNext();) {
        tanarEsHallgatok = (Map.Entry) i.next();
        tanar = (String) tanarEsHallgatok.getKey();
        hallgatok = (HashSet) tanarEsHallgatok.getValue();
        if (hallgatok.size() == letszam) {
            out.println(tanar);
        }
    }
}
}

```

A megoldás 5 lépésből áll:

1. Beolvassuk a tanárok neveit és a név alapján mindenkihez hozzárendelünk egy kollekciót, amibe a hallgatók nevei kerülnek majd. A hallgatók kollekciója bármilyen kollekció lehetne, mi a `java.util.HashSet`-et választottuk. Minden tanárhoz egy ilyen kollekciót rendelünk és a tanár nevét használjuk kulcsként. Mivel kulcs alapú elérés kell a legcélszerűbb egy táblázat (`Map`) használata. A leggyakrabban használt ilyen kollekció a `java.util.Hashtable`, ezért a `tanarokHallgatoEntrySet` statikus attribútum ilyen típusú lesz.
2. Beolvassuk a második állomány sorait és a '#' mentén két részre vágjuk. Így kapjuk a hallgatók és tanáraik nevét. A tanár neve alapján lekérjük a hallgatók kollekcióját és ebbe beszurjuk a hallgatót.
3. Megállapítjuk, hogy mennyi a legnagyobb hallgatói létszám, amivel egy oktató rendelkezik. Ehhez minden oktató esetén a hallgatók listájának méretét használjuk fel.
4. A legmagasabb létszámú hallgatóval rendelkező tanárok neveit kiírjuk. Ehhez a `printTanarByHallgatoLetszam` metódust használjuk, mivel az az 5. lépésben is felhasználható lesz. Itt lényegében a kulcsokon és a hallgatói listákon együtt kell végighaladni. Ez a hashtábla bejegyzéseinek (`Map.Entry`) lekérdezésével történik. (Lehetne úgy csinálni, hogy csak a kulcsokat soroljuk fel és minden kulcshoz lekérjük a hallgatók a kollekcióját, így elkerülhetjük a `Map.Entry` interfészt. Ez a megoldás azonban kevésbé hatékony.)
5. A 0 db hallgatóval rendelkező tanárokat a 4. lépéshez teljesen hasonlóan kell felsorolni.

2. megoldás

Ahhoz, hogy a legtöbb hallgatóval rendelkező ill. hallgatók nélküli tanárokat megkeressük, elég a tanárokhöz a létszámot tárolni. Így most az adatszerkezetünk egy olyan táblázat, ahol a kulcs a tanár neve az érték pedig csak a tanár hallgatóinak létszáma (nem pedig a hallgatók neveinek listája).

A táblázatot megvalósíthatjuk két szinkronban indexelt tömbbel, ahol az első a tanárok nevei tartalmazza, míg a második a létszámokat. Mivel azonban az első mérete nem tervezhető, használjunk a tanárok neveinél egy `Vector`-t.

```

import java.io.*;
import java.util.*;

```

```

public class TanarHallgato2 {
    private static Vector tanarok;
    private static int[] letszamok;

    public static void main(String[] args) {
        tanarok = new Vector();
        FileReader in = null;
        try {
            readTanarok(new BufferedReader(in = new FileReader(args[0])));
            letszamok = new int[tanarok.size()];
            in = new FileReader(args[1]);
            readHallgatok(new BufferedReader(in));
        } catch (IOException e) {
            System.err.println("IO Hiba."
                + " Ellenőrizze a megadott állományokat!");
        } finally {
            try { in.close(); } catch (Exception e) {}
        }
        int maxLetszam = getMaxLetszam();
        System.out.println("Legtöbb hallgatóval (" + maxLetszam
            + ") rendelkező tanárok:");
        System.out.println("~~~~~");
        printTanarByHallgatoLetszam(maxLetszam, System.out);
        System.out.println();
        System.out.println("Hallgató nélküli tanárok:");
        System.out.println("~~~~~");
        printTanarByHallgatoLetszam(0, System.out);
    }

    private static void readTanarok(BufferedReader in) throws IOException {
        String s;
        while ((s = in.readLine()) != null) {
            tanarok.add(s);
        }
    }

    private static void readHallgatok(BufferedReader in)
        throws IOException {
        String s;
        String tanar;
        while ((s = in.readLine()) != null) {
            tanar = s.substring(s.indexOf("#") + 1);
            letszamok[tanarok.indexOf(tanar)]++;
        }
    }

    private static int getMaxLetszam() {
        int max = 0;
        for (int i = 0; i < letszamok.length; i++) {
            if (letszamok[i] > max) {
                max = letszamok[i];
            }
        }
        return max;
    }

    private static void printTanarByHallgatoLetszam(int letszam,
        PrintStream out) {
        for (int i = 0; i < letszamok.length; i++) {
            if (letszamok[i] == letszam) {

```

```

        out.println(tanarok.get(i));
    }
}
}

```

Itt is 5 fő lépés van:

1. Beolvassuk a tanárok neveit a `tanarok` `Vector`-ba.
2. Mivel ismerjük a tanárok számát egy ekkora `letszamok` nevű `int` tömböt hozunk létre az egyes tanárok hallgatói létszámának tárolására. (A tömb minden eleme 0-ra inicializálódik.) Ezután beolvassuk a második állomány sorait és a '#' utáni részt tekintjük a tanár nevének. Ez alapján meghatározzuk a tanár helyét a `Vector`-ban és így a párhuzamosan indexelt `letszamok` tömb megfelelő elemét növeljük.
3. Megállapítjuk, hogy mennyi a legnagyobb hallgatói létszám. Ez egy `int` tömbbeli `maximumelem` keresés.
4. Megkeressük a legmagasabb létszámok indexét és ez alapján a `tanarok` `Vector` megfelelő elemét, azaz a tanár nevét kiírjuk.
5. A 0 db hallgatóval rendelkező tanárokat a 4. lépéshez teljesen hasonlóan kell felsorolni.

Megjegyzés:

- A 2. megoldás sokkal egyszerűbb, mert sokkal egyszerűbb adatszerkezetet használ. Viszont éppen ezért csak ilyen egyszerű kérdések megválaszolására alkalmas. Az 1. megoldásban használt adatszerkezet könnyebben adaptálható hasonló, de nehezebb feladatokra.

2. Feladat

Írjon STATIKUS METÓDUST, mely egy paraméterként kapott állományból, amely pontosan egy Java nyelven megírt osztály forráskódját tartalmazza, kigyűjti és elhelyezi egy szöveges állomány soraiban az osztályban lévő konstruktor(ok) formális paraméterlistáját.

A kód nem tartalmaz megjegyzést, sztring- és karakterkonstanst.

Megoldás

A feladat érdekes, de a kezdő olvasónak nem feltétlen ajánljuk. Tipikusan az a feladat, amit az ember egy dolgozat/verseny során a végére hagy.

A feladat a Java nyelv szintaxisának ismeretét kéri számon. A feladat nem egyszerű, tudnunk kell hogyan találhatjuk meg a konstruktorok definícióját. A feladatot könnyebben és nehezebben is érthetjük aszerint, hogy a „pontosan egy Java nyelven megírt osztály forráskódját tartalmazza” szövegbe az esetleges belső és lokális osztályokat beleértjük-e. A mi megoldásunk ezekkel is számol.

A „Java nyelven megírt” viszont azt jelenti, hogy az állományban szereplő forrás szintaktikailag helyes (különben nem lenne Java nyelven megírva!).

A konstruktor neve megegyezik az osztály nevével. Így akár az állomány nevéből is kideríthető lenne, ha eltekintenénk a belső osztályoktól. Így azonban mondjuk azt, hogy az osztály neve a `class` kulcsszó után közvetlenül biztosan szerepel. (Ez alól kivétel a névtelen osztály, de annak meg nem lehet konstruktora, szerencsénkre.)

Konstruktorok definíciója az osztály törzsében legkülső szinten helyezhető el, vagyis a `class` utáni első nyitó '{' és az azt záró '}' között. A zárójelpárok megtalálása számlálással egyszerű feladat.

Az osztály törzsében az osztály neve még a legkülső szinten sem biztos, hogy konstruktort jelöl. Hol fordulhat még elő:

- Változódeklarációban lehet attribútum típusa.

- Kifejezésben (pl. kezdőértékkadás) lehet példányosítás, vagy
- típuskonverzió, utóbbi mindig zárójelek közt van.
- Formális paraméter típusaként is zárójelek közt van.
- Ha az osztály kivételosztály, akkor metódusok és konstruktorok `throws` részében is lehet.
- Típusként akár tömbtípus alaptípusa is lehet és így tömbkonstrukcióban is szerepelhet.
- Jelölhet visszatérési típust, ilyenkor még egy azonosító (a metódus neve) áll utána.
- Reméljük, hogy ennyi.

A zárójelek közt elhelyezett eseteket kiszűrhetjük azzal, hogy ezeket a kapcsolószárójelekhez hasonlóan kezeljük, s így ilyenkor már nem lesz az osztálynév legkülső szinten.

A konstruktornál azt is észrevehetjük, hogy az osztály neve után közvetlen (esetleges üres karakterekkel elválasztva) nyitózároljel jön. Ilyen másik eset csak a példányosítás marad, de ott meg az osztály neve előtt közvetlen a `new` kulcsszó szerepel, vagy egy `'.'`-ra (pontra) végződő token.

Utóbbi azért kell, mert a minősített neveknél használt pont és az azonosítók közt üres karakterek lehetnek

```
pl.: new java .      util .      Hashtable()    // nem konstruktor
```

Megjegyzés és sztringkonstans nincs az állományban (ettől ugyanis a feladat nem lenne sokkal érdekesebb, csak bonyolultabb és időigényesebb).

Összefoglalva, amit keresünk az osztály törzsében az először a `class` kulcsszó, innen megvan az osztály neve. Ezután az ide tartozó törzsben keressük legkülső szinten az osztálynév olyan előfordulásait, amit `'('` követ és nem előz meg a `new` kulcsszó. Ezeknél az előfordulásoknál az osztály neve után álló `'('` és `')'` közti részt keressük.

A mi megoldásunk a konstruktorokat módosítok és `throws` rész nélkül, de az osztálynévvel és a zárójelpárral együtt adjuk meg. Pl.: `Hashtable(int initialCapacity, float loadFactor)`

Ezt azért tesszük, hogy a paraméter nélküli konstruktor is jól látható legyen.

A szöveg feldarabolását az állomány teljes tartalmának egy `String`-be történő beolvasása után a `StringTokenizer` osztállyal végezzük. A tokenizálást üres karakterek és zárójelek mentén hajtjuk végre, úgy hogy ezeket is tokeneknek tekintjük. A tokenizálás közben számoljuk, hogy hányadik karakternél járunk. Így miután egy osztály konstruktoraival végeztünk az osztály fejléce előtti részt elhagyva újabb osztályok után (belső és lokális) kutathatunk, amíg találunk.

```
import java.io.*;
import java.util.*;

public class KonstruktorKereso {
    private static final String URES_KARAKTEREK = " \t\n\r\f";

    private static int poz;

    public static void konstruktortKeres(String inFile, String outFile) {
        Reader in = null;
        PrintWriter out = null;
        try {
            in = new FileReader(inFile);
            String inStr = readFile(in);
            out = new PrintWriter(new FileWriter(outFile));
            int classBodyPoz = 0;
            while (true) {
                classBodyPoz = konstruktortKeres(inStr, out);
                if (classBodyPoz == -1) break;
            }
        }
    }
}
```

```

        inStr = inStr.substring(classBodyPoz);
    }
} catch (IOException e) {
    System.err.println("IO Hiba");
} finally {
    try {in.close(); } catch (Exception e) {}
    try {out.close(); } catch (Exception e) {}
}
}

private static String readFile(Reader in) throws IOException {
    StringBuffer sb = new StringBuffer();
    int c;
    while ((c = in.read()) != -1) {
        sb.append((char) c);
    }
    return sb.toString();
}

private static String barmilyenToken(StringTokenizer st) {
    String token = st.nextToken();
    poz += token.length();
    return token;
}

private static String nemUresToken(StringTokenizer st) {
    String token;
    while (st.hasMoreTokens()) {
        token = barmilyenToken(st);
        if (URES_KARAKTEREK.indexOf(token.charAt(0)) < 0) {
            return token;
        }
    }
    return null;
}

private static int konstruktortKeres(String inStr, PrintWriter out) {
    String className;
    poz = 0;
    int bodyPoz;
    String token = null;
    StringTokenizer st = new StringTokenizer(inStr,
                                             URES_KARAKTEREK + "{}();",
                                             true);

    while (true) {
        token = nemUresToken(st);
        if (token == null) return -1;
        if ("class".equals(token)) break;
    }
    className = nemUresToken(st);

    System.out.println("class:" + className);

    while (!"{}".equals(token)) {
        token = nemUresToken(st);
    }
    bodyPoz = poz;
    konstruktortKeres(className, st, out);

    return bodyPoz;
}

```



```

    }

    private static boolean isNyitozarojel(String token) {
        return "{" .equals(token) || "(" .equals(token);
    }

    private static boolean isZarozarojel(String token) {
        return "}" .equals(token) || ")" .equals(token);
    }

    private static void konstruktortKeres(String className,
                                         StringTokenizer st,
                                         PrintWriter out) {

        int zarojel = 1;
        String token = "";
        String prevToken;
        int konstruktorok = 0;
        while (zarojel > 0) {
            // st-nek még van elem, mert az inpu szabályos.
            prevToken = token;
            token = nemUresToken(st);
            if (zarojel == 1 && className.equals(token)) {
                String elozoToken = prevToken;
                prevToken = token;
                token = nemUresToken(st);
                if ("(" .equals(token) && !"new" .equals(elozoToken)
                    && !elozoToken.endsWith(".")) {
                    konstruktorok++;
                    out.print(className);
                    while (!")" .equals(token)) {
                        out.print(token);
                        token = barmilyenToken(st);
                    }
                    out.println(")");
                    out.flush();
                }
            } else if (isNyitozarojel(token)) {
                zarojel++;
            } else if (isZarozarojel(token)) {
                zarojel--;
            }
        }
        if (konstruktorok == 0) {
            out.println(className + "()");
            out.flush();
        }
    }
}

```

A megoldást egy programban felhasználva a `java.util.Hashtable` osztály megjegyzésektől megtisztított forrásán futtattuk. Az eredmény:

```

Hashtable(int initialCapacity, float loadFactor)
Hashtable(int initialCapacity)
Hashtable()
Hashtable(Map t)
KeySet()
EntrySet()
ValueCollection()
Entry(int hash, Object key, Object value, Entry next)
Enumerator(int type, boolean iterator)

```

```
EmptyEnumerator()
EmptyIterator()
```

Megjegyzések:

- A feladatban lehet hogy mi sem gondoltunk minden eshetőségre. A dolgozatban sem vártunk tökéletes megoldásokat, csak azt, hogy néhány esetre gondoljanak a hallgatók.
- A mintaillesztések `indexOf`-fal nehezen végezhetőek lennének, mert így részzavakat is megtalálnánk.
- A feladat talán egyszerűsíthető, ha valaki ügyesen le tudja írni az illesztendő mintákat reguláris kifejezésekkel.
- Használható lenne a `java.io.StreamTokenizer` osztály is, de egy java forrás talán elfér a memóriában.
- A konstruktor paraméterlistáját ugyanolyan tördeléssel írjuk ki, ahogy a forrásban van.

3. Feladat

Készítsen STATIKUS METÓDUST, amely paraméterben megkap egy szabályos S-kifejezést tartalmazó sztringet.

Pl.:

(+ (* 2 -3.2 (- 5 0.4) 4) (* -2.4 4) 5.56 7 10)

Az S-kifejezés a `+` `-` `*` `/` függvényneveket (operátorokat) és valós számokat, mint aktuális paramétereket (operandusokat) tartalmaz. A statikus metódus visszatérési értéként adja vissza az S-kifejezés értékét.

A `+` függvény visszatérési értéke a paramétereinek (kettő vagy több) összege, a `*`-é a szorzatuk. A `-` és a `/` pontosan két operandussal rendelkezhet.

Megoldás

A megoldás egy rekurzív algoritmus lesz, amiben a `java.util.StringTokenizer` osztályt használjuk az elemekre bontáshoz.

```
import java.util.StringTokenizer;

public class SKiertekelo {
    private static final String URES_KARAKTEREK = " \t\n\r\f";

    public static double kiertekel(String skif) {
        StringTokenizer skifTokenizer =
            new StringTokenizer(skif, "(" + URES_KARAKTEREK, true);
        return kiertekel(null, skifTokenizer);
    }

    private static String elsoNemUresToken(StringTokenizer skif) {
        String token;
        while (skif.hasMoreTokens()) {
            token = skif.nextToken();
            if (URES_KARAKTEREK.indexOf(token.charAt(0)) < 0) {
                return token;
            }
        }
        return null;
    }

    private static double kiertekel(String firstToken,
                                    StringTokenizer skif) {
        String token =
            firstToken != null
```

```

        ? firstToken
        : elsoNemUresToken(skif);

// token != null, mert akkor nem lenne
// a paraméter helyes S-kifejezes
if ("(".equals(token)) {
    token = elsoNemUresToken(skif);
    char op = token.charAt(0);
    double ertek = kiertekel(null, skif);
    double kovParam;
    while (!")".equals(token = elsoNemUresToken(skif))) {
        kovParam = kiertekel(token, skif);
        switch (op) {
            case '*':
                ertek *= kovParam;
                break;
            case '+':
                ertek += kovParam;
                break;
            case '/':
                ertek /= kovParam;
                break;
            case '-':
                ertek -= kovParam;
                break;
        }
    }
    return ertek;
} else {
    // nincs zárójel az elején, ez egy ATOM -> valós szám
    return Double.parseDouble(token);
}
}
}

```

A záró zárójelek megtalálásához előre kell tekintenünk egy tokennel. Ha viszont mégsem záró zárójelet olvasunk, akkor azt vissza kellene rakni a `StringTokenizer`-be. Ezt nem lehet, ezért paraméterként adjuk át a meghívott függvénynek.

Megjegyzések:

- A kivonás és az osztás is rendelkezhet több paraméterrel, ezt nem vizsgáljuk. Ilyenkor több kivonás ill. osztás történik, ahogy LISP-ben is.
- Zárójelezett ATOM nincs. Pl.: (5), ez az 5 függvény hívását jelentené LISP-ben.

4. Feladat (módosított változat)

Írjon `OSZTÁLY`-t, amely megvalósítja a halmaz adatszerkezetet, implementálva a következő interfészt:

```

import java.util.Iterator;

interface Halmaz{
    void hozzáad(Object o);
    boolean benneVan(Object o);
    Halmaz metszet(Halmaz h);
    Halmaz unio(Halmaz h);
    Halmaz minusz(Halmaz h);
    Halmaz descartesSzorzat(Halmaz h);
    Iterator iterator();
}

```

A hozzáad függvény új elemeket helyez el a halmazban. A benneVan teszti, hogy az adott elem eleme-e a halmaznak. A metszet, unio, minusz es descartesSzorzat műveletek értelemszerűen egy új halmazt adnak vissza. Az iterator() metódus a halmaz elemeit felsoroló objektumot ad vissza.

Megoldás

Több megoldást adunk. Az első használja az API egy meglevő kollekcióját, a második tömbbel valósítja meg a halmazt. A harmadikban a feladat egy csöppet módosított változatát oldjuk meg.

A Descartes-szorzat eredménye elempárok halmaza. Egy elempárt egy 2 elemű Object tömb reprezentál. (Másik lehetőség az lenne, ha egy külön osztályt írnánk erre a célra, pl. RendezettPar névvel.)

1. megoldás

```
import java.util.*;

public class HalmazImpl extends HashSet implements Halmaz {

    public void hozzáad(Object o) {
        add(o);
    }
    public boolean benneVan(Object o) {
        return contains(o);
    }
    public Halmaz metszet(Halmaz h) {
        Halmaz h2 = new HalmazImpl();
        Object o;
        for (Iterator i = iterator(); i.hasNext(); ) {
            o = i.next();
            if (h.benneVan(o)) {
                h2. hozzáad(o);
            }
        }
        return h2;
    }

    public Halmaz unio(Halmaz h) {
        Halmaz h2 = new HalmazImpl();
        for (Iterator i = iterator(); i.hasNext(); ) {
            h2. hozzáad(i.next());
        }
        for (Iterator i = h.iterator(); i.hasNext(); ) {
            h2. hozzáad(i.next());
        }
        return h2;
    }

    public Halmaz minusz(Halmaz h) {
        Halmaz h2 = new HalmazImpl();
        Object o;
        for (Iterator i = iterator(); i.hasNext(); ) {
            o = i.next();
            if (!h.benneVan(o)) {
                h2. hozzáad(o);
            }
        }
        return h2;
    }
}
```

```

public Halmaz descartesSzorzat(Halmaz h) {
    Halmaz h2 = new HalmazImpl();
    Object o1, o2;
    for (Iterator i = iterator(); i.hasNext(); ) {
        o1 = i.next();
        for (Iterator j = h.iterator(); j.hasNext(); ) {
            o2 = j.next();
            h2.hozzaad(new Object[] {o1, o2});
        }
    }
    return h2;
}

/*
public Iterator iterator() {
    return super.iterator();
}
*/
}

```

A `HashSet` osztály rendelkezik a szükséges funkcionalitással, csak a megfelelő interfészhez kell illeszteni. Az `iterator()` metódust változtatás nélkül használhatjuk (megjegyzésekben szerepel).

2. megoldás

```

import java.util.Iterator;
import java.util.NoSuchElementException;

public class HalmazImpl2 implements Halmaz {

    private static final int KEZDO_MERET = 50;
    private static final int MERET_NOVEKMENY = 50;
    private Object[] elemek = new Object[KEZDO_MERET];
    private int elemSzam = 0;

    private void novelHaKell() {
        if (elemSzam == elemek.length) {
            Object[] ujElemek = new Object[elemSzam + MERET_NOVEKMENY];
            System.arraycopy(elemek, 0, ujElemek, 0, elemSzam);
            elemek = ujElemek;
        }
    }

    public void hozzaad(Object o) {
        if (!benneVan(o)) {
            novelHaKell();
            elemek[elemSzam++] = o;
        }
    }

    public boolean benneVan(Object o) {
        for (int j = 0; j < elemSzam; j++) {
            if (elemek[j].equals(o)) return true;
        }
        return false;
    }

    public Halmaz metszet(Halmaz h) {
        Halmaz h2 = new HalmazImpl2();
        Object o;
        for (Iterator i = iterator(); i.hasNext(); ) {

```

```

        o = i.next();
        if (h.benneVan(o)) {
            h2.hozzaad(o);
        }
    }
    return h2;
}

public Halmaz unio(Halmaz h) {
    Halmaz h2 = new HalmazImpl2();
    for (Iterator i = iterator(); i.hasNext();) {
        h2.hozzaad(i.next());
    }
    for (Iterator i = h.iterator(); i.hasNext();) {
        h2.hozzaad(i.next());
    }
    return h2;
}

public Halmaz minusz(Halmaz h) {
    Halmaz h2 = new HalmazImpl2();
    Object o;
    for (Iterator i = iterator(); i.hasNext();) {
        o = i.next();
        if (!h.benneVan(o)) {
            h2.hozzaad(o);
        }
    }
    return h2;
}

public Halmaz descartesSzorzat(Halmaz h) {
    Halmaz h2 = new HalmazImpl2();
    Object o1, o2;
    for (Iterator i = iterator(); i.hasNext();) {
        o1 = i.next();
        for (Iterator j = h.iterator(); j.hasNext();) {
            o2 = j.next();
            h2.hozzaad(new Object[]{o1, o2});
        }
    }
    return h2;
}

public Iterator iterator() {
    return new Iterator() {
        int poz = 0;

        public boolean hasNext() {
            return poz < elemSzam;
        }

        public Object next() throws NoSuchElementException {
            if (!hasNext()) {
                throw new NoSuchElementException();
            }
            return elemek[poz++];
        }

        public void remove() throws IllegalArgumentException {
            throw new IllegalArgumentException();
        }
    };
}

```

```

    }
    };
}

```

A 2. megoldás csak az alpműveletek megvalósításában különbözik az elsőtől.

Az `iterator()` metódusban névtelen osztályt használtunk. Lehetett volna különálló, belső vagy lokális osztályt is használni. Hasonló a feladat a 2003/04-es év 2. feladatsorának 3. feladata, ahol ezzel részletesen foglalkozunk. (Ott is `Itertor`-t kell megvalósítani). Szintén hasonló az említett feladatsor 2. feladata, de ott sokkal egyszerűbb az interfész.

3. megoldás

A ZH-n feladott eredeti feladatból hiányzott az `iterator()` metódus.

```

interface Halmaz2{
    void add(Object o);
    boolean benneVan(Object o);
    Halmaz2 metszet(Halmaz2 h);
    Halmaz2 unio(Halmaz2 h);
    Halmaz2 minusz(Halmaz2 h);
    Halmaz2 descartesSzorzat(Halmaz2 h);
}

```

Így kevesebb metódust kell implementálnunk, de a feladat veszít általánosságából, mert az `unio(...)` és a `descartesSzorzat(...)` műveletek csak a paraméterül kapott halmaz elemeinek felsorolásával valósítható meg.

Ezt a paraméter halmaza reprezentációjára, tényleges dinamikus típusára tett megszorítással tehetjük. Feltételezzük, hogy a paraméter dinamikus típusa megegyezik az implementáló osztályéval és explicit típuskonverzióval elérjük a paraméter belső reprezentációját.

Az egyszerűség kedvéért a reprezentáció egy `HashSet` típusú attribútummal történik (most nem öröklünk, mint az 1. megoldásban).

```

import java.util.*;

public class Halmaz2Impl implements Halmaz2 {

    private HashSet halmaz = new HashSet();

    public void add(Object o) {
        halmaz.add(o);
    }

    public boolean benneVan(Object o) {
        return halmaz.contains(o);
    }

    public Halmaz2 metszet(Halmaz2 h) {
        Halmaz2 h2 = new Halmaz2Impl();
        Object o;
        for (Iterator i = halmaz.iterator(); i.hasNext(); ) {
            o = i.next();
            if (h.benneVan(o)) {
                h2.add(o);
            }
        }
        return h2;
    }

    public Halmaz2 unio(Halmaz2 h) {
        Halmaz2 h2 = new Halmaz2Impl();

```

```

        for (Iterator i = halmaz.iterator(); i.hasNext(); ) {
            h2.add(i.next());
        }
        // Csak akkor tudom felsorolni a paraméter elemeit,
        // ha ismerem a szerkezetét, vagyis ugyanilyen szerkeztű.
        // Explicit konverziót alkalmazunk.
        for (Iterator i = ((Halmaz2Impl) h).halmaz.iterator();
            i.hasNext(); ) {
            h2.add(i.next());
        }
        return h2;
    }

    public Halmaz2 minusz(Halmaz2 h) {
        Halmaz2 h2 = new Halmaz2Impl();
        Object o;
        for (Iterator i = halmaz.iterator(); i.hasNext(); ) {
            o = i.next();
            if (!h.benneVan(o)) {
                h2.add(o);
            }
        }
        return h2;
    }

    public Halmaz2 descartesSzorzat(Halmaz2 h) {
        Halmaz2 h2 = new Halmaz2Impl();
        Object o1, o2;
        for (Iterator i = halmaz.iterator(); i.hasNext(); ) {
            o1 = i.next();
            // Csak akkor tudom felsorolni a paraméter elemeit,
            // ha ismerem a szerkezetét, vagyis ugyanilyen szerkeztű.
            // Explicit konverziót alkalmazunk.
            for (Iterator j = ((Halmaz2Impl) h).halmaz.iterator();
                j.hasNext(); ) {
                o2 = j.next();
                h2.add(new Object[] {o1, o2});
            }
        }
        return h2;
    }
}

```

2003/04 I. félév

1. Feladatsor

1. Feladat

```

class Komplex {
    protected final double re, im;

    public Komplex(double re_, double im_) {
        re = re_; im = im_;
        System.out.println("Új komplex szám: " + toString());
    }

    public Komplex plusz(Komplex k) {
        System.out.println("K + K: " + toString() + " + "
            + k.toString());
        return new Komplex(re + k.re, im + k.im);
    }
}

```



```

    }
    public String toString() {
        if(im >= 0)
            return "(" + re + " + " + im + "i)";
        else
            return "(" + re + " - " + (-im) + "i)";
    }
}

class Valós extends Komplex {
    public Valós(double r) {
        super(r, 0);
        System.out.println("Új valós szám: " + toString());
    }

    public Komplex plusz(Komplex k) {
        System.out.println("V + K: " + toString() + " + "
            + k.toString());
        return new Komplex(re + k.re, k.im);
    }

    public Valós plusz(Valós v) {
        System.out.println("V + V: " + toString() + " + "
            + v.toString());
        return new Valós(re + v.re);
    }
    public String toString() {
        return "(" + re + ")";
    }
}

public class Próba {
    public static void main(String args[]) {
        Komplex kk = new Komplex(3.4, -5.6);
        Komplex kv = new Valós(1.2);
        Valós vv = new Valós(7.8);

        System.out.println(kk.plusz(kv).toString());
        System.out.println(vv.plusz(kv).toString());
        System.out.println(kv.plusz(kv).toString());
        System.out.println(vv.plusz(kv).plusz(kk).toString());
        kk = vv;
        System.out.println(kk.plusz(new Valós(9)).toString());
    }
}

```

- a) Adja meg a program kimenetét! Jelölje, mely utasítás mely sorokat eredményezte!
- b) A 0.8 valós számhoz hozzá szeretnénk adni a kv változó által hivatkozott objektumot. Hogyan járhatunk el, ha az eredményt egy Valós típusú változóba szeretnénk elhelyezni?

Megoldás

a)

Kimenetet a kód `System.out.println(...)` sorai eredményeznek. Azt kell figyelnünk, hogy a program futása során ezekre mikor, milyen paraméterekkel kerül a vezérlés. A kimenetet a főprogram (nem üres) soraihoz kapcsolódva adjuk meg:

1. sor

```
Komplex kk = new Komplex(3.4, -5.6);
```

```
Új komplex szám: (3.4 - 5.6i)
```

2. sor

```
Komplex kv = new Valós(1.2);
```

```
Új komplex szám: (1.2)
Új valós szám: (1.2)
```

Egy öröklött osztály példányosításakor a szülő osztályok konstruktorai is meghívásra kerülnek az öröklődési láncon végighaladva az `java.lang.Object` konstruktorával kezdve. Ezért itt a komplex konstruktor is eredményez kimenetet.

A komplex konstruktor az új példány kiírásához annak `toString()` példánymetódusát hívja meg, ahol a rendszer dinamikus kötést alkalmaz. Így a `Valós` osztálybeli `toString()` változat kerül meghívásra. Ezért szerepel az eredményben `(1.2)` és nem `(1.2, 0.0)` !

3. sor

```
Valós vv = new Valós(7.8);
```

```
Új komplex szám: (7.8)
Új valós szám: (7.8)
```

Az előző sorral megegyező eset.

4. sor

```
System.out.println(kk.plusz(kv).toString());
```

```
K + K: (3.4 - 5.6i) + (1.2)
Új komplex szám: (4.6 - 5.6i)
(4.6 - 5.6i)
```

A `kk` változó `plusz` metódusa hívódik meg először (`K + K`), ahol egy új `Komplex` szám is létrejön (-> konstruktorhívás). Végül a visszaadott `Komplex` számot `String`-gé alakítva kiírjuk.

5. sor

```
System.out.println(vv.plusz(kv).toString());
```

```
V + K: (7.8) + (1.2)
Új komplex szám: (9.0 + 0.0i)
(9.0 + 0.0i)
```

A `vv` változó `plusz` metódusa hívódik meg. De itt két eset lehetséges vagy `plusz(Valós)`, vagy a `plusz(Komplex)` változat hívódhat meg. Ezt az dönti el, hogy mi a paraméter *deklarált* típusa. S mivel `kv` deklarált típusa `Komplex`, ezért a `plusz(Komplex)` hívódik meg (`V + K`). Itt még egy új `Komplex` szám is létrejön (-> konstruktorhívás). Végül a visszaadott `Komplex` számot `String`-gé alakítva kiírjuk.

6. sor

```
System.out.println(kv.plusz(kv).toString());
```

```
V + K: (1.2) + (1.2)
Új komplex szám: (2.4 + 0.0i)
(2.4 + 0.0i)
```

A `kv` változó `plusz` metódusa hívódik meg. Mivel `kv` deklarált típusa `Komplex`, ezért az egyedüli meghívható metódus a `plusz(Komplex)` szignatúrájú. Viszont `kv` dinamikus típusa `Valós` és dinamikus kötést kell használni a metódus hívásánál, ezért a `Valós` típusban

deklarált (felülírt) metódus kerül meghívásra ($v + k$). Itt még egy új `Komplex` szám is létrejön (-> konstruktorhívás). Végül a visszaadott `Komplex` számot `String`-gé alakítva kiírjuk.

7. sor

```
System.out.println(vv.plusz(kv).plusz(kk).toString());

V + K: (7.8) + (1.2)
Új komplex szám: (9.0 + 0.0i)
K + K: (9.0 + 0.0i) + (3.4 - 5.6i)
Új komplex szám: (12.4 - 5.6i)
(12.4 - 5.6i)
```

A `vv.plusz(kv)` eredményezi az első két sort, az 5. sornál leírtak szerint. Ennek az eredménye az ott leírtak szerint `Komplex` típusú objektum, aminek egyetlen `plusz` metódusa van, s az hívódik meg ($K + K$). Itt még egy új `Komplex` szám is létrejön (-> konstruktorhívás). Végül a visszaadott `Komplex` számot `String`-gé alakítva kiírjuk.

8. sor

```
kk = vv;
```

Ez a sor nem eredményez kimenetet, de ezután a `kk` által hivatkozott objektum egy `Valós` lesz, így a deklarált és a dinamikus típus eltér (mint `kv` esetén), ezért a metódusok dinamikus kötésére jobban oda kell majd figyelniük.

9. sor

```
System.out.println(kk.plusz(new Valós(9)).toString());

Új komplex szám: (9.0)
Új valós szám: (9.0)
V + K: (7.8) + (9.0)
Új komplex szám: (16.8 + 0.0i)
(16.8 + 0.0i)
```

A `plusz` argumentumának példányosítása eredményezi az első két sort (mint 2. sorban). Ezután eldől, hogy melyik `plusz` hívódik meg. Mivel `kk` *deklarált* típusa `Komplex` ezért a szignatúra csak `plusz(Komplex)` lehet. De a dinamikus kötés miatt a `Valós`-ban deklarált változat hívódik meg ($v + k$). Az eredmény megint egy új `Komplex` szám lesz (-> konstruktorhívás), amely számot `String`-gé alakítva kiírunk.

b)

Mivel `Valós` típusú eredményt szeretnénk, ezért azt az egyetlen `plusz` metódust kell meghívni, amely visszatérési értéke `Valós`. Ez a `Valós` típusban van deklaráva. Ennek meghívása csak úgy lehetséges, ha egy `Valós` típusú referenciát felhasználva végezzük el a hívást és az átadott paraméter típusa is `Valós` (nem elég, ha a dinamikus típus `Valós` csak). Ahhoz hogy `kv` típusát a dinamikus típusával egyező `Valós` típusúvá tegyük explicit típuskonverziót kell használnunk.

Két lehetséges megoldás:

1. `Valós v = new Valós(0.8).plusz((Valós) kv);`
2. `Valós v = ((Valós) kv).plusz(new Valós(0.8));`

2. Feladat

```
public class MunkaCsoport {
    public static final int MAX_DOLGOZÓ = 10;

    private String név;
```

```

        private double összFizetés = 0;
        private Dolgozó[] dolgozók = new Dolgozó[MAX_DOLGOZÓ];

        public MunkaCsoport(String név) {
            this.név = név;
        }

        public void újDolgozó(String név, double fizetés) {
            /*
            Ha a csoportban még nem dolgozik MAX_DOLGOZÓ
            dolgozó, akkor vegye fel az új dolgozót.
            Figyeljen az összFizetés karbantartására.
            */
        }

        /* package */ void számoljÖsszFizetést() {
            /*
            Ez a művelet tartja karban az összFizetés értékét a
            csoportban dolgozók fizetésének megfelelően.
            */
        }

        public double getÖsszFizetés() {
            // Adj vissza az összfizetést!
        }
    }

    public class Dolgozó {
        private String név;
        private double fizetés;
        private MunkaCsoport csoport;

        public Dolgozó(/* szükséges paraméterek */) {
            /* A dolgozó adatainak inicializálása */
        }

        public void setFizetés(double fizetés) {
            /*
            Figyeljen a MunkaCsoport összesített fizetésének
            karbantartására is.
            */
        }

        public double getFizetés() {
            /* Adj vissza a fizetést! */
        }
    }
}

```

- a) Egészítse ki a kódot a megjegyzéseknek megfelelően!
- b) Egészítse ki a konstruktort! Legyen a dolgozóknak egy azonosítójuk (egész), ami automatikusan kap értéket a konstruktorban. Minden következő azonosító az előzőtől tízzel nagyobb, a legkisebb 100 legyen.

Megoldás

```

public class MunkaCsoport {
    public static final int MAX_DOLGOZÓ = 10;

    private String név;
    private double összFizetés = 0;
    private Dolgozó[] dolgozók = new Dolgozó[MAX_DOLGOZÓ];

```

```

private int letszam;

public MunkaCsoport(String név) {
    this.név = név;
}

public void újDolgozó(String név, double fizetés) {
    if(létszám < MAX_DOLGOZÓ){
        dolgozók[létszám++] = new Dolgozó(név, fizetés, this);
        számoljÖsszfizetést(); // vagy összFizetés += fizetés;
    }
}

/* package */ void számoljÖsszfizetést() {
    összFizetés = 0;
    for (int i = 0; i < létszám; ++i)
        összFizetés += dolgozók[i].getFizetés();
}

public double getÖsszfizetés() {
    return összFizetés();
}
}

public class Dolgozó {
    private String név;
    private double fizetés;
    private MunkaCsoport csoport;

    private static int nextId = 100;
    private int id;

    public Dolgozó(String n, double f, MunkaCsoport mcs) {
        név = n;
        fizetés = f;
        csoport = mcs;
        id = nextId++;
    }

    public void setFizetés(double fizetés) {
        this.fizetés = fizetés;
        csoport.számoljÖsszfizetést();
    }

    public double getFizetés() {
        return fizetés;
    }
}

```

a)

Az újDolgozó metódusban az aktuális példányt át kell adni paraméterként a this segítségével:

```
dolgozók[létszám++] = new Dolgozó(név, fizetés, this);
```

A setFizetés metódusban vissza kell hívni a csoport egy metódusát:

```
csoport.számoljÖsszfizetést();
```

b)

A megoldáshoz osztályváltozót kell használni.

2. Feladatsor

1. Feladat

Írjon statikus metódust, amely paraméterül kap két sztringet, és visszaadja egy `Integer` objektumban, hányszor fordul elő a rövidebb a hosszabban.

Megoldás

Itt több jó megoldás is adható. Mi hármat adunk, melyek mindegyike a `String` osztály valamely metódusát használja (nem írjuk meg két sztring karaktereinek összehasonlítását feleslegesen).

1. megoldás

```
public static Integer occurrences1(String s1, String s2) {
    int count = 0;
    if (s1.length() < s2.length()) {
        String tmp = s1; s1 = s2; s2 = tmp;
    }
    for (int i = 0; i < s1.length()-s2.length()+1; i++) {
        if (s1.regionMatches(i, s2, 0, s2.length())) count++;
    }
    return new Integer(count);
}
```

2. megoldás

```
public static Integer occurrences2(String s1, String s2) {
    int count = 0;
    if (s1.length() < s2.length()) {
        String tmp = s1; s1 = s2; s2 = tmp;
    }
    int i = s1.indexOf(s2);
    while (i >= 0) {
        count++;
        i = s1.indexOf(s2, i+1);
    }
    return new Integer(count);
}
```

3. megoldás

```
public static Integer occurrences3(String s1, String s2) {
    int count = 0;
    if (s1.length() < s2.length()) {
        String tmp = s1; s1 = s2; s2 = tmp;
    }
    for (int i=0; i<s1.length()-s2.length()+1; i++) {
        if (s1.substring(i, i+s2.length()).equals(s2)) count++;
    }
    return new Integer(count);
}
```

A három megoldás közül a második a legrövidebb, mivel kevesebb ciklusiterációból áll, mint az első és nem hoz létre felesleges objektumokat, mint a harmadikban a `substring` hívás.

2. Feladat

Adott a bináris fákat reprezentáló `BinárisFa` interfész:

```
interface BinárisFa {
    BinárisFa getBal();
    BinárisFa getJobb();
    int getÉrték();
}
```

- a) Írjon statikus metódust, amely paraméterül kap egy bináris fát, és visszaadja a fában található számok összegét. A `getÉrték` metódus az adott fa gyökerében tárolt értéket adja vissza. Az üres fát a null érték reprezentálja.
- b) Adott a következő interfész:

```
interface Halmaz {
    void hozzáad(int i);
    boolean tartalmaz(int i);
}
```

Készítsen osztályt, amely értelemszerűen implementálja az interfészt. Adjon meg az osztályban egy olyan konstruktort, amelynek egyetlen paramétere egy bináris fa. A konstruktor töltsen fel a halmazt a fa elemeivel!

Megoldás

a)

A feladat ezen része azt kéri számon, hogy a hallgató ismeri-e az interfészek használatát. Itt nem kell a `BinárisFa` interfészt implementálni.

```
public static int összeg(BinárisFa f) {
    if (f == null) return 0;
    return f.getÉrték() + összeg(f.getBal()) + összeg(f.getJobb());
}
```

b)

Itt igazából két feladat van. Implementálni kell a `Halmaz` interfészt, ez többféleképpen is megoldható. A másik feladat a megfelelő konstruktor megadása, itt lényegében egyféle megközelítés képzelhető el.

A `Halmaz` implementáció miatt több megoldást adunk itt meg, mivel azonban a konstruktor ugyanaz, azt csak az első megoldás tartalmazza.

A `Halmaz` implementációja során két dolog különösen fontos:

- Implementálni egy osztályt azt jelenti, hogy az interfész-metódusok jól működjenek. Ehhez más osztályok felhasználhatók.
- Értelemszerűen implementálni egy halmazt azt is jelenti, hogy a halmaz egy elemet csak egyszer tartalmazhat.

b) - 1. megoldás

```
import java.util.HashSet;

public class HalmazImpl implements Halmaz {
    private HashSet elemek = new HashSet();

    public HalmazImpl(BinárisFa f) {
        hozzáad(f);
    }

    public void hozzáad(BinárisFa f) {
        if (f == null) return;
        hozzáad(f.getÉrték());
        hozzáad(f.getBal());
        hozzáad(f.getJobb());
    }
}
```

```

    }

    public void hozzáad(int i) {
        elemek.add(new Integer(i));
    }

    public boolean tartalmaz(int i) {
        return elemek.contains(new Integer(i));
    }
}

```

A legkézenfekvőbb dolog az, ha a Java kollekciói közül választunk egy megfelelőt. A legmegfelelőbb valamilyen `Set`, azaz vagy a `HashSet` vagy a `TreeSet` osztályt választhatjuk. Azonban ezek az osztályok nem képesek egyszerű típusok értékeinek közvetlen tárolására. Ezért használjuk az `Integer` csomagoló osztályt. Azt is megjegyezhetjük, hogy a `Set` osztályok a tartalmazott elemeket az `equals` metódussal hasonlítják össze, így kiszűrjük az elemek duplikációját.

b) - 2. megoldás

```

import java.util.HashSet;

public class HalmazImpl extends HashSet implements Halmaz {

    /* A konstruktor és hozzáad(BinárisFa f)
       metódus megegyezik az 1. megoldásbelivel. */

    public void hozzáad(int i) {
        add(new Integer(i));
    }

    public boolean tartalmaz(int i) {
        return contains(new Integer(i));
    }
}

```

Az újrafelhasználás nem csak egy attribútum használatával (kompozícióval) valósítható meg, hanem öröklődéssel. Így még rövidebb, elegánsabb kódot kapunk, viszont a `HalmazImpl` osztály most jóval bővebb metóduskészlettel is rendelkezik, hiszen maga is egy `HashSet`.

b) - 3. megoldás

```

import java.util.Vector;

public class HalmazImpl implements Halmaz {
    private Vector elemek = new Vector();

    /* A konstruktor és hozzáad(BinárisFa f)
       metódus megegyezik az 1. megoldásbelivel. */

    public void hozzáad(int i) {
        Integer iObj = new Integer(i);
        if (!elemek.contains(iObj)) {
            elemek.add(iObj);
        }
    }

    public boolean tartalmaz(int i) {
        return elemek.contains(new Integer(i));
    }
}

```



```
}
```

Aki nem ismeri a kollekciókat elég jól, az talán nem gondol a `HashSet` osztályra. Természetesen használható valami szokásosabb, ismerősebb osztály is például a `Vector` osztály, amit valószínűleg már használtunk. Itt azonban nekünk kell az elemek duplikációját megelőzni a `hozzáad` metódusban.

Megjegyzés:

- Itt is használhattunk volna öröklődést.

b) - 4. megoldás

```
public class HalmazImpl3 implements Halmaz {
    private static final int KEZDŐ_MÉRET = 50;
    private static final int MÉRET_NÖVEKMÉNY = 50;
    private int[] elemek = new int[KEZDŐ_MÉRET];
    private int elemSzám = 0;

    /* A konstruktor és hozzáad(BinárisFa f)
       metódus megegyezik az 1. megoldásbelivel. */

    private void növelHaKell() {
        if (elemSzám == elemek.length) {
            int[] újElemek = new int[elemSzám + MÉRET_NÖVEKMÉNY];
            System.arraycopy(elemek, 0, újElemek, 0, elemSzám);
            elemek = újElemek;
        }
    }

    public void hozzáad(int i) {
        if (!tartalmaz(i)) {
            növelHaKell();
            elemek[elemSzám++] = i;
        }
    }

    public boolean tartalmaz(int i) {
        for (int j=0; j<elemSzám; j++) {
            if (elemek[j] == i) return true;
        }
        return false;
    }
}
```

Végezetül megadtunk egy a kollekciók használata nélküli megoldást. Ez egy egyszerű tömbös megoldás, ahol az elemeket tároló tömb az igényeknek megfelelően dinamikus növekszik.

3. Feladat

Adott a következő két Java osztály:

```
import java.util.Iterator;

public class GyakCsoport {
    private static final int MAX_LÉTSZÁM = 18;
    private String tantárgykód;
    private int csoportSorszám;
    private Hallgató[] hallgatók = new Hallgató[MAX_LÉTSZÁM];
    private int létszám = 0;

    public GyakCsoport(String tantárgykód, int csoportSorszám)
    {
```

```

        this.tantárgykód = tantárgykód;
        this.csoportSorszám = csoportSorszám;
    }

    public void addHallgató(Hallgató h) {
        hallgatók[létszám++] = h;
    }

    public int getLétszám() {
        return létszám;
    }

    public Iterator iterator() {
        // ...
    }
}

```

```

public class Hallgató {
    private String név;
    private String szak;
    private int évfolyam;

    public Hallgató(String név, String szak, int évfolyam) {
        this.név = név;
        this.szak = szak;
        this.évfolyam = évfolyam;
    }
    // ...
}

```

- Implementálja a GyakCsoport osztály `iterator()` metódusát úgy, hogy a kapott `Iterator` a csoporthoz tartozó hallgatókat adja vissza!
- Adja meg a `CsoportMegteltException` osztályt ellenőrzött kivételként. Javítsa ki az `addHallgató` metódust úgy, hogy az kivételt dobjon hiba esetén.
- Írjon programot (`Beoszt.java`), amely parancssori paraméterként megkapja másodéves ("II.") PTM-es ("PTM") hallgatók neveit, és ezen hallgatókat megfelelő számú Programozás 2 ("I1205") gyakorlati csoportban helyezi el. A csoportokat 1-től sorszámozza! A csoportokat lehetőség szerint teljesen fel kell tölteni. Kezelje a `CsoportMegteltException` kivételt!
Mivel a `MAX_LÉTSZÁM` mező privát láthatóságú, a csoport megteltét csak a kiváltott kivétel jelezheti.

Megoldás

a)

A feladatnak több, lényegét tekintve is különböző megoldását adhatjuk meg.

Lényegileg kétféle megoldást adunk:

- Használjuk a Java API egy kollekcióját, amit feltöltünk a csoport elemeivel, majd a kollekció iterátora lesz az eredmény.
- Létrehozunk egy osztályt, amely implementálja az `Iterator` interfészt és ezt példányosítjuk. Az új osztály lehet független (külső), belső, lokális vagy névtelen osztály. A természetes Javás megoldás a névtelen osztály használata, de megadjuk majd egy független osztállyal történő megvalósítást is.

a) - 1. megoldás

```

import java.util.Vector;
import java.util.Iterator;

```

```

public class GyakCsoport {
    ...
    public Iterator iterator() {
        Vector v = new Vector();
        for (int i = 0; i < létszám; i++) {
            v.add(hallgatók[i]);
        }
        return v.iterator();
    }
}

```

Ez a megoldás rövid és frappáns egy ZH esetén. Valódi projekt esetén azonban költséges lehet ha egy tömb tartalmát minden iteráció alkalmával átmásoljuk egy vektorba!

a) - 2. megoldás

```

import java.util.NoSuchElementException;
import java.util.Iterator;

public class GyakCsoport {
    ...

    private void töröl(int index) {
        System.arraycopy(hallgatók, index+1,
                        hallgatók, index, --létszám-index);
    }

    public Iterator iterator() {
        return new Iterator() {

            private int pos = 0;
            private int last = -1;

            public boolean hasNext() {
                return pos < létszám;
            }

            public Object next() {
                if (hasNext()) {
                    last = pos;
                    return hallgatók[pos++];
                } else {
                    throw new NoSuchElementException();
                }
            }

            public void remove() {
                if (last < 0) throw new IllegalStateException();
                töröl(last);
                last = -1;
                pos--;
            }
        };
    } // iterator()

} // class

```

Ez a megoldás teljes, abban az értelemben, hogy még a törlés művelete is meg van valósítva. A gyakorlatban a törlést nem mindig valósítjuk meg, így a feladat a `remove` üres megvalósítással is elfogadható.

A `remove` szokásos üres megvalósítása vagy üres blokkal `{}` történik, vagy a megfelelő nem ellenőrzött kivétel kiváltásával:

```
public void remove() {  
    throw new UnsupportedOperationException();  
}
```

Megjegyzések:

- A törlés fenti megvalósítása nem szinkronizált, így több párhuzamosan használt iterátor esetén problémák adódhatnak (nem „threadsafe”).
- Üres `remove` esetén nincs szükség `töröl` metódusra a `GyakCsoport` osztályban.

a) - 3. megoldás

Az `Iterator` interfészt külső osztályban is implementálhatjuk. A felsorolandó objektum adatait az új osztály konstruktorában vehetjük át. Az adat lehetne a felsorolandó `GyakCsoport` példány, de mivel annak privát tagjaira, így a hallgató tömbre sem hivatkozhatunk az iterátorosztályból, válasszuk paraméternek a felsorolandó tömböt és a létszámot.

Tehát valójában egy tömböt kell felsorolnunk, adott elemszámig. A felsorolandó tömb típusa lehet `Hallgató[]`, viszont mivel az `Iterator` `next` metódusa mindenképpen `Object`-et ad vissza, lehet a paraméter `Object[]` is, így az új iterátorosztály másféle tömb felsorolására is alkalmas lesz, az újrafelhasználhatósága jelentősen nő.

TömbIterator:

```
import java.util.NoSuchElementException;  
import java.util.Iterator;  
  
public class TömbIterator implements Iterator {  
  
    private Object[] t;  
    private int méret;  
    private int pos = 0;  
  
    public TömbIterator(Object[] t, int méret) {  
        this.t = t;  
        this.méret = méret;  
    }  
  
    public boolean hasNext() {  
        return pos < méret;  
    }  
  
    public Object next() {  
        if (hasNext()) {  
            return t[pos++];  
        } else {  
            throw new NoSuchElementException();  
        }  
    }  
  
    public void remove() {  
        throw new UnsupportedOperationException();  
    }  
}
```

GyakCsoport:

```
import java.util.Iterator;
```

```
public class GyakCsoport {
    ...

    public Iterator iterator() {
        return new TömbIterator(hallgatók, létszám);
    }
}
```

Megjegyzések

- Az általános tömbfelsoroló azért használható, mert teljesül, hogy egy `Hallgató[]` referencia, pl. a `hallgatók` esetén igaz, hogy `hallgatók instanceof Object[]`.
- A `TömbIterator` újrafelhasználhatósága növelhető további más paraméterű konstruktorok megadásával. Pl.: csak (tömb) vagy (tömb, tól, ig)

b)

CsoportMegteltException:

```
public class CsoportMegteltException extends Exception {
    public CsoportMegteltException() {}
    public CsoportMegteltException(String message) { super(message); }
}
```

GyakCsoport:

```
import java.util.Iterator;

public class GyakCsoport {
    ...

    public void addHallgató(Hallgató h) throws CsoportMegteltException {
        if (létszám == MAX_LÉTSZÁM) {
            throw new CsoportMegteltException();
        }
        hallgatók[létszám++] = h;
    }
}
```

A `CsoportMegteltException` osztályban nem szükséges megadni konstruktorokat elfogadható a következő is:

```
public class CsoportMegteltException extends Exception {}
```

c)

Nagyon sok jó megoldás képzelhető el. Fontos, hogy nem használhatjuk sem a nevesített konstans, sem a `18` literált, valamint az, hogy az `addHallgató` *minden* hívását (ha egynél többször fordul elő) megfelelő `try - catch` blokkal vegyük körül. Gyakori hiba lehet az is, hogy azt a hallgatót, akinél az előző csoport betelik, elfelejtjük a következő csoportba beletenni, átugrik rajta a ciklus.

A feladat szövege nem specifikálja, hogy mi legyen a létrehozott `GyakCsoportok`-kal, mi ezeket egy `Vector`-ban helyezzük el.

c) - 1. megoldás

```
import java.util.Vector;

public class Beoszt {

    public static void main(String[] args) {
        Vector gyakorlatok = new Vector();
        GyakCsoport gy;
        for (int csoportSzám = 1, i = 0;
            i < args.length;
```

```

        csoportSzám++) {

        gy = new GyakCsoport("I1205", csoportSzám);
        gyakorlatok.add(gy);
        try {
            for (; i < args.length; i++) {
                gy.addHallgató(new Hallgató(args[i], "PTM", 2));
            }
        } catch (CsoportMegteltException ex) {}
    }
}

} // class

```

A három megoldás közül a legszebb (a szerző szerint).

c) - 2. megoldás

```

import java.util.Vector;

public class Beoszt {

    public static void main(String[] args) {
        Vector gyakorlatok = new Vector();
        GyakCsoport gy;
        int csoportSzám = 1;
        gyakorlatok.add(gy = new GyakCsoport("I1205", csoportSzám++));
        for (int i = 0; i < args.length; i++) {
            try {
                gy.addHallgató(new Hallgató(args[i], "PTM", 2));
            } catch (CsoportMegteltException ex) {
                gy = new GyakCsoport("I1205", csoportSzám++)
                gyakorlatok.add(gy);
                i--;
            }
        }
    }
}

```

A kivételkezelőben csökkentjük az *i* számláló értékét, hogy a túlszorduló hallgató ne maradjon ki.

c) - 3. megoldás

```

import java.util.Vector;

public class Beoszt {

    public static void main(String[] args) {
        Vector gyakorlatok = new Vector();
        GyakCsoport gy;
        int csoportSzám = 1;
        gyakorlatok.add(gy = new GyakCsoport("I1205", csoportSzám++));
        Hallgató h = null;

        for (int i = 0; i < args.length; i++) {
            try {
                h = new Hallgató(args[i], "PTM", 2);
                gy.addHallgató(h);
            } catch (CsoportMegteltException ex) {
                gy = new GyakCsoport("I1205", csoportSzám++)
                gyakorlatok.add(gy);
            }
        }
    }
}

```

```

        try {
            gy.addHallgató(h);
        } catch (CsoportMegteltException ex2) {
            // csak MAX_LÉTSZÁM == 0 esetén következhetne be.
        }
    }
}
}

```

A kivételkezelőben adjuk hozzá a túlsorduló hallgatót az új gyakorlati csoporthoz, ezért az ott szereplő `addHallgató` hívás köré is `try - catch` blokkot kell írunk.

3. Feladatsor

1. Feladat

Írjon osztályt, amely implementálja az `Iterator` interfészt. Az osztály konstruktorában megkap egy `Vector[]` típusú paramétert. Az osztály - mint iterátor - sorolja fel a paraméterként kapott vektorok valamennyi elemét! A `next()` metódus dobjon `NoSuchElementException`-t, ha nincs következő elem, a `remove()` metódus pedig dobjon `UnsupportedOperationException`-t.

Megoldás

Ez a feladat is többféleképpen oldható meg. Mi itt három különböző gondolatmenetet mutatunk be. Mindegyiknél apró buktató lehet, ha a konstruktor paramétertömbjében nem kezeljük az esetleges üres vektorokat és `null` tömbelemeket. Ez gyakori pontatlanság, de nem súlyos hiba. Mivel a feladatban megadott kiváltandó kivételek mind a `RuntimeException` leszármazottai, nem kötelező deklarálnunk a metódusok fejében.

1. megoldás

Ebben a megoldásban a két pozíciószámlálót használunk. Az első (`vPos`) a vektortömbön belüli, a második (`pos`) az aktuális vektoron belüli pozíciót jelöli. A léptetést elosztottan valósítjuk meg: a `next()` metódus mindig meghívja a `hasNext()` metódust, amely elvégzi az esetleg szükséges léptetést a vektortömbön belül, valamint jelzi, hogy a léptetés eredményeként „túlmentünk-e” az utolsó elemen. Az aktuális tömbön belüli pozíciónövelés a `next()` metódusban történik.

```
import java.util.*;
```

```
public class VectorArrayIterator1 implements Iterator {
```

```
    private Vector[] v;
    private int vPos = 0;
    private int pos = 0;
```

```
    public VectorArrayIterator1(Vector[] v) {
        this.v = v;
    }
```

```
    public boolean hasNext() {
        if (vPos >= v.length)
            return false;
        else if (v[vPos] != null && v[vPos].size() > pos)
            return true;
        else {
            pos = 0;

```

```

        do {
            vPos++;
        } while (vPos < v.length
                && (v[vPos] == null || v[vPos].size() == 0));
        return vPos < v.length;
    }
}

public Object next() throws NoSuchElementException {
    if (!hasNext()) {
        throw new NoSuchElementException();
    }
    return v[vPos].get(pos++);
}

public void remove() throws UnsupportedOperationException {
    throw new UnsupportedOperationException();
}
}

```

2. megoldás

Ebben a megoldásban egy pozíciószámlálót (`vPos`) a vektortömbön belüli pozíció jelölésére. Az aktuális vektor felsorolását annak egy saját iterátorával (`it`) végezzük el. A léptetést elosztottan valósítjuk meg: a `next()` metódus mindig meghívja a `hasNext()` metódust, amely elvégzi az esetleg szükséges léptetést a vektortömbön belül és inicializálja az aktuális tömböt felsoroló iterátort (`it`), valamint jelzi, hogy a léptetés eredményeként „túlmentünk-e” az utolsó elemen. Az aktuális iterátor léptetése automatikusan megtörténik az `it.next()` híváskor.

```

import java.util.*;

public class VectorArrayIterator2 implements Iterator {

    private Vector[] v;
    private int vPos = 0;
    private Iterator it = null;

    public VectorArrayIterator2(Vector[] v) {
        this.v = v;
    }

    public boolean hasNext() {
        if (it != null && it.hasNext())
            return true;
        else if (vPos >= v.length)
            return false;
        else {
            do {
                if (v[vPos] == null) {
                    it = null;
                } else {
                    it = v[vPos].iterator();
                }
                vPos++;
            } while ((it == null || !it.hasNext()) && vPos < v.length);
            return it.hasNext();
        }
    }

    public Object next() throws NoSuchElementException {

```



```

        if (!hasNext()) {
            throw new NoSuchElementException();
        }
        return it.next();
    }

    public void remove() throws UnsupportedOperationException {
        throw new UnsupportedOperationException();
    }
}

```

3. megoldás

A végére hagytuk a legkönnyebben programozható ám legnagyobb tár- és időigényű változatot. Itt a megoldás lényegében a konstruktorban történik, ahol a paraméterben található valamennyi nem null vektor tartalmát egy saját tömbben tároljuk el. Egy vektor minden elemének hozzáadását a `Collection` interfészbeli `addAll(...)` metódus is megkönnyíti. Ez után már csak annyi dolgunk marad, hogy az összes elemet tartalmazó új vektort felsoroljuk az iterátorával. (Magára a vektorra a továbbiakban nincs is szükség, ezért lehet ez lokális változó.)

```

import java.util.*;

public class VectorArrayIterator3 implements Iterator {

    private Iterator it;

    public VectorArrayIterator3(Vector[] v) {
        Vector vAll = new Vector();
        for (int i = 0; i < v.length; i++) {
            if (v[i] != null) {
                vAll.addAll(v[i]);
            }
        }
        it = vAll.iterator();
    }

    public boolean hasNext() {
        return it.hasNext();
    }

    public Object next() throws NoSuchElementException {
        if (!hasNext()) {
            throw new NoSuchElementException();
        }
        return it.next();
    }

    public void remove() throws UnsupportedOperationException {
        throw new UnsupportedOperationException();
    }
}

```

2. Feladat

Adott egy szöveges állomány (autok.txt), amelyben egy autóraktár készletét tartjuk nyilván. Az állomány sorai a következő alakúak:

```
típus:szín:gyártási év
```

Írjon programot, amely létrehoz egy szöveges állományt (osszesit.txt), és típusonként, azon belül színenként összesítve írja ki az állományba az autókat.

Formátuma soronként:

típus:szín:darabszám

Megoldás

A feladatban meg kell tudni a nyitni a bemenet ill. később a kimenet szöveges állományát. A bemenetet soronként dolgozzuk fel és minden sort az utolsó (vagy második) kettőspont mentén kettévágunk. A kettévágott sor eleje lesz a számunkra fontos, az egyezőket kell összeszámolni. Az összeszámolást legcélszerűbben egy kulcs szerint címezhető táblázatban oldhatjuk meg, ahol a kulcsot a bemenet sorainak első része adja. Ilyen táblázat a `Hashtable`, ami azonban nem képes primitív típusok kezelésére. Ezért létrehozunk egy saját `Számláló` osztályt, amely példányaiban egy-egy `int` értéket növelgetünk. A beolvasás és feldolgozás után már csak az eredményt kell kiírni a kimeneti állományba. Ezt a `Hashtable` kulcsainak iterálásával és az azokhoz tartozó számlálók értékének kiíratásával tesszük meg.

Fontos, hogy az I/O műveletek kivételeit megfelelő módon kezeljük a programban.

```
import java.util.*;
import java.io.*;

public class Autok {

    private static class Számláló {
        int érték = 0;
        public String toString() {
            return String.valueOf(érték);
        }
    }

    private static Hashtable autoSzámláló;

    public static void main(String[] args) {
        FileReader in = null;
        FileWriter out = null;
        try {
            in = new FileReader("autok.txt");
            összesít(in);
            in.close();
            out = new FileWriter("osszesit.txt");
            kiír(out);
            out.close();
        } catch (IOException e) {
            System.err.println("Váratlan IO hiba: " + e.getMessage());
            System.err.println("Ellenőrizze a megadott paramétereket!");
        } finally {
            try { in.close(); } catch (Exception e) {}
            try { out.close(); } catch (Exception e) {}
        }
    }

    private static void összesít(Reader inReader) throws IOException {
        BufferedReader in = new BufferedReader(inReader);
        autoSzámláló = new Hashtable();
        String s;
        Számláló sz;
        while ((s = in.readLine()) != null) {
            // Az utolsó ':' előtti rész lesz a kulcs.
```

```

        s = s.substring(0, s.lastIndexOf(':'));
        sz = (Számláló) autoSzámláló.get(s);
        if (sz == null) {
            sz = new Számláló();
            autoSzámláló.put(s, sz);
        }
        sz.érték++;
    }
}

private static void kiír(Writer outWriter) {
    PrintWriter out = new PrintWriter(outWriter);
    Object s;
    for (Iterator i = autoSzámláló.keySet().iterator();
         i.hasNext(); ) {
        s = i.next();
        out.println(s + ":" + autoSzámláló.get(s));
    }
}
}

```

Megjegyzések:

- A számlálás megvalósításához használhatnánk a beépített `Integer` csomagoló osztályt is, de ennek az értéke nem módosítható, csak lekérdezhető. Így egy növelés megvalósítása új `Integer` példányosítást igényelne, ami nem hatékony.
- A `Hashtable` iterálása hatékonyabb lenne, ha nem kulcs szerint iterálnánk, hanem táblázat bejegyzéseit az `autoSzámláló.entrySet().iterator()` hívással, amely iterátor `Map.Entry` elemeket ad vissza. (Egy `Hashtable` ilyen módon történő bejárására példát a 2002/03 I. félév 3. feladatsor 1. feladatának 1. megoldása tartalmaz.)
- A leszámlálás nem csak `Hashtable`-lel valósítható meg. Másik megoldás lenne például, ha az autóberegysések számára hozunk létre külön osztályt és ez tartalmazza mind a kulcs, mind a számláló attribútumot. Ezután ilyen bejegyzéseket már `Vector`-ban vagy tömbben tárolhatunk, és az eltárolt bejegyzések közt a szokásos módon kereshetünk. Ez a megoldás persze jóval lassabban fut (kivéve ha esetleg rendezést is megvalósítunk) és kevésbé használja ki a java API-t.

3. Feladat

Írjon osztályt, amely konstruktorában paraméterként megkap egy `Reader`-t. A `Reader` által olvasható adatfolyam sorai megjegyzést vagy kulcs-érték párt tartalmazhatnak a következő formában:

```

#megjegyzés

vagy

kulcs=érték

```

Az osztály rendelkezzeik egy `public String get(String kulcs)` metódussal, amely visszaadja az adott kulcshoz tartozó értéket.

Definiálja és váltsa ki a `KulcsÜtközésException` illetve `NincsIlyenKulcsException` kivételeket arra az esetre, ha egy kulcs többször szerepel az állományban, vagy ha egy keresett kulcs nem létezik az állományban!

A megjegyzéseket az osztály hagyja figyelmen kívül!

Megoldás

A feladatban egyszerűen egy táblázatot kell implementálnunk. Ehhez felhasználhatjuk a Java API-ban egyébként is már meglévő kollekcíósztályokat. Három megoldást adunk melyek mindegyike a konstruktorban olvassa be az adatokat és váltja ki a `KulcsÜtközésException` kivételt. Ezután a memóriában lévő adatok közül keressük ki a megfelelőt a `get(...)` metódusban.

Mivel a kivételosztályok definíciója megegyezik ezért azokat adjuk meg először, majd a három kicsit különböző táblázatot.

Kivételosztályok:

```
public class KulcsÜtközésException extends Exception {
    public KulcsÜtközésException() {}
    public KulcsÜtközésException(String msg) { super(msg); }
}

public class NincsIlyenKulcsException extends Exception {
    public NincsIlyenKulcsException() {}
    public NincsIlyenKulcsException(String msg) { super (msg); }
}
```

1. megoldás

A legkézenfekvőbb megoldás a mindenki által jól ismert `Hashtable` használata attribútumként. A `get(...)` metódusban figyelünk a megfelelő típuskonverzió elvégzésére.

```
import java.util.*;
import java.io.*;

public class Táblázat1 {

    private Hashtable h;

    public Táblázat1(Reader input) throws IOException,
        KulcsÜtközésException {
        BufferedReader in = new BufferedReader(input);
        h = new Hashtable();
        String s, kulcs;
        int ePos;
        while ((s = in.readLine()) != null) {
            if (s.startsWith("#")) continue;
            ePos = s.indexOf("=");
            kulcs = s.substring(0, ePos);
            if (h.containsKey(kulcs)) {
                throw new KulcsÜtközésException("Duplikált kulcs: "
                    + kulcs);
            }
            h.put(kulcs, s.substring(ePos+1));
        }

        public String get(String kulcs) throws NincsIlyenKulcsException {
            String rv = (String) h.get(kulcs);
            if (rv == null) throw new NincsIlyenKulcsException();
            return rv;
        }
    }
}
```

2. megoldás

`Hashtable` helyett felhasználható még a belőle származó `Properties` osztály is, így nem kell a típuskonverzióval sem bajlódni. A változatosság kedvéért most öröklődést használunk.

```

import java.util.*;
import java.io.*;

public class Táblázat2 extends Properties {

    public Táblázat2(Reader input) throws IOException,
        KulcsÜtközésException {
        BufferedReader in = new BufferedReader(input);
        String s, kulcs;
        int ePos;
        while ((s = in.readLine()) != null) {
            if (s.startsWith("#")) continue;
            ePos = s.indexOf("=");
            kulcs = s.substring(0, ePos);
            if (containsKey(kulcs)) {
                throw new KulcsÜtközésException("Duplikált kulcs: "
                    + kulcs);
            }
            setProperty(kulcs, s.substring(ePos+1));
        }

        public String get(String kulcs) throws NincsIlyenKulcsException {
            String rv = super.getProperty(kulcs);
            if (rv == null) throw new NincsIlyenKulcsException();
            return rv;
        }
    }
}

```

Megjegyzés:

- Ha a feladat nem kérné a kulcsütközések figyelését, akkor a `Properties` osztály `load(...)` metódusa is használható lenne, tovább egyszerűsítve a feladatot.

3. megoldás

Ez a megoldás egy sokkal egyszerűbb adattípust, `Vector`-t használ. Két párhuzamosan indexelt vektorral lényegében egy táblázatot valósítunk meg.

```

import java.util.*;
import java.io.*;

public class Táblázat3 extends Properties {

    private Vector kulcsok;
    private Vector értékek;

    public Táblázat3(Reader input) throws IOException,
        KulcsÜtközésException {
        BufferedReader in = new BufferedReader(input);
        kulcsok = new Vector();
        értékek = new Vector();
        String s, kulcs;
        int ePos;
        while ((s = in.readLine()) != null) {
            if (s.startsWith("#")) continue;
            ePos = s.indexOf("=");
            kulcs = s.substring(0, ePos);
            if (kulcsok.contains(kulcs)) {
                throw new KulcsÜtközésException("Duplikált kulcs: "
                    + kulcs);
            }
            kulcsok.add(kulcs);
        }
    }
}

```

```

        értékek.add(s.substring(ePos+1));
    }
}

public String get(String kulcs) throws NincsIlyenKulcsException {
    int index = kulcsok.indexOf(kulcs);
    if (index == -1) throw new NincsIlyenKulcsException();
    return (String) értékek.get(index);
}
}

```

Megjegyzés:

- A `Vector` alternatívájaként tömböt is használhatnánk, így semmilyen kollekcíóra nem lenne szükség a `java.util` csomagból. Ekkor viszont a dinamikus méretezést, valamint a `contains(...)` és az `indexOf(...)` hívásoknak megfelelő kódot nekünk kell implementálni.

Ajánlott irodalom, hivatkozások

- [Prog2] Programozás 2 ZH-sorok, http://dragon.unideb.hu/~gabora/prog2_zh_sorok/
- [Java13] Nyékiné G. Judit szerk., *Java 2 Referencia 1.3*, ELTE TTK Hallgatói Alapítvány, Budapest, 2001
- [Bruce] Bruce Eckel, *Thinking in Java*, Prentice Hall PTR; 1st Edition 1998, 2nd Edition 2000, 3rd Edition 2002, <http://www.bruceeckel.com/>,
<http://www.mindview.net/Books>
- [JSun] Sun Java Technology honlap, <http://java.sun.com/>
- [JDoc] Java Technology dokumentációk, <http://java.sun.com/docs/>
- [JTut] The Java Tutorial, <http://java.sun.com/docs/books/tutorial/index.html>