BÉKÉSI JÓZSEF, DÁVID BALÁZS, HAJDU LÁSZLÓ

OBJEKTUMORIENTÁLT PROGRAMOZÁS PÉLDATÁR

INFORMATIKUS FELSŐOKTATÁSI SZAKKÉPZÉSI SZAKOSOKNAK

Lektorálta: Tóth Attila

Jelen tananyag a Szegedi Tudományegyetemen készült az Európai Unió támogatásával. Projekt azonosító: EFOP-3.4.3-16-2016-00014

A feladatgyűjtemény a Tufte-Style Book latex sablon segítségével készült: HTTP://WWW.LATEXTEMPLATES.COM/TEMPLATE/TUFTE-STYLE-BOOK

Tartalomjegyzék

Bevezető

A példatár az Objektumorientált programozás című tantárgy szemináriumának anyagához kapcsolódó gyakorló feladatokat és azok megoldását tartalmazza. A tantárgy előfeltétele a programozás alapjainak ismerete, így feltételezzük, hogy a hallgatók ekkor már tisztában vannak a legalapvetőbb programozási módszerekkel, ismerik a változók, adattípusok, kifejezések, azonosítók, vezérlési szerkezetek és metódusok fogalmát. Ugyancsak feltételezzük alapvető összetettebb adatstruktúrák, mint például a tömb ismeretét. A programozás technikáinak elsajátításához fontos az algoritmikus gondolkodás megtanulása. Véleményünk szerint ez elsősorban úgy lehetséges, hogy sok feladat megoldásán keresztül fejlesztjük ezt a képességünket. Ehhez kívánunk segítséget nyújtani ezzel a feladatgyűjteménnyel. A feladatokat témák szerint csoportosítottuk. Az első részben a vezérlési szerkezetek, majd a metódusok használatához kapcsolódó feladatok találhatók. Ezek megoldásához még nem szükséges az objektumorientált technikák használata, így azok számára is hasznos lehet, akik csak a programozás alapjaival ismerkednek. A következő fejezetek már az objektumorientált módszerekhez kapcsolódnak, követve az egyszerűbbtől a bonyolultabbak felé történő haladás elvét. Először egyszerűbb objektumok megvalósítását igénylő feladatok, majd az objektumok metódusaihoz kapcslódó példák is találhatók. A példatár második részében már komplexebb feladatok vannak, amelyek egymáshoz szorosan kapcsolódó kisebb részfeladatokból állnak és általában a teljes megoldás egy összetettebb program. Ezek a feladatok általában az objektumorientált programozás fejlettebb technológiáinak - mint például az öröklődés - ismeretét is igénylik. Minden feladat után megtalálható a megoldás Java nyelven megírt kód formájában, így a hallgatók közvetlen módon ellenőrizhetik saját megoldásukat. Reményeink szerint a példatár segítséget nyújt a hallgatóknak a programozási ismereteik elmélyítésében. A programozási ismeretek alapvető kompetenciái közül a példatár különösen fejlesztheti az alkalmazásfejlesztés és tesztelés módszertanának elsajátítását, a legalapvetőbb feladat-megoldási elvek, módszerek, a fő szoftverfejlesztési paradigmák, alapvető programozási módszertanok használatának technikáit. Fejlesztheti az informatikai feladatok megoldásához szükséges együttműködésre, egyéni és team munkában való hatékony munkavégzésre való képességet. A következőkben részletezzük, hogy az egyes fejezetek hogyan járulnak hozzá a tantárggyal kapcsolatosan megfogalmazott kompetenciák fejlesztéséhez. Az első fejezet a vezérlési szerekezetekkel kapcsolatos feladatokat tartalmazza. Ezek a feladatok hozzásegítik a hallgatót, hogy rendelkezzen az információrendszerekkel, adatbázisokkal és programozással kapcsolatos alapismeretekkel. Így képes lesz vállalati, üzleti folyamatokhoz kapcsolódó egyszerű tervezési, programozási feladatok elvégzésére. Megsmeri a legfontosabb vezérlőszerkezeteket, így a szelekciókat és az iterációkat, a megoldásokban szelekciós és iterációs utasításokat alkalmaz. A

feladatok hozzásegítik ezek pontos megvalósításához, így képes lesz ezek önálló használatára. A második fejezet fő témája a metódusok, amelyeket nevezhetünk elemi informatikai rendszerkomponenseknek is. Az itt szereplő feladatokat megoldva a hallgató megismeri az informatika alapvető technikáit, az informatikai rendszerkomponensek szerepét, feladatát és működését. Így képes és nyitott lesz az új informatikai technológiák, programnyelvek és módszerek megismerésére. Gyakorolhatja az önálló és csapatmunkát is, megismeri ezek korlátait. A feladatok megoldása által a hallgató megismeri a metódusokat, a metódusok felépítését, azok lokális változóinak szerepét, ezeket alkalmazni tudja, a pontos és önálló munkára való képessége fejlődik. A harmadik fejezet speciálisan a rekurzív metódusokkal foglalkozik. A feladatok segítségével a hallgató megismeri a rekurziót, a megoldások során rekurzív metódusokat alkalmaz, igyekszik ezeket pontosan használni és be kell tartania a rekurzív metódushívás szabályait. A negyedik fejezet az egyszerű objektumokra vezető feladatokat tartalmaz.Ezen feladatok által a hallgató megismeri a fejlesztési módszertanok szerepét, és egy fontos módszertant használ. Képes lesz objektumorientált programok implementációjára legalább egy programnyelven és fejlesztési környezetben. Ezáltal képes lesz szakmai együttműködésben rendszertervezési, -fejlesztési részfeladatok elvégzésére, dokumentálására. Így munkájáért felelősséget kell vállalnia önálló feladatvégzésnél és csoportmunkában egyaránt. A feladatok megoldásával a hallgató tisztában lesz az osztályok valamint az objektumok használatával, készítésével (referenciatípus fogalma, objektum létrehozása, deklarálása). Munkája során egyszerűbb osztályokat és objektumokat készít, objektumokat deklarál, törekszik ezek precíz használatára, a megoldások segítségével munkáját önállóan tudja ellenőrizni. Az ötödik fejezet az objektumok metódusaihoz kapcsolódó feladatokból áll. Ezek megoldásával a hallgató megismeri a felhasználói követelmények feltárásának és elemzésének alapvető technikáit, módszereit. Képes lesz szakmai vélemény kialakítására a szoftverfejlesztéshez szükséges technológiák, hardver és szoftver eszközök kiválasztása során.Ezáltal nyitott lesz a képesítésével, szakterületével kapcsolatos szakmai, technológiai, fejlesztési eredmények megismerésére, befogadására, és törekszik saját tudásának megosztására. Önálló lesz a munkájához kapcsolódó előírások betartásában és a vonatkozó dokumentumok elkészítésében. A feladatokon keresztül a hallgató megismeri az osztály fogalmát, felépítését, deklarációit, valamint az osztálytag és példánytag fogalmát. A megoldás során osztály- és példánytagokat tartalmazó osztályokat hoz létre. Ezáltal nyitott lesz az objektumok különböző módszerekkel történő létrehozására, az objektumok különböző célú alkalmazására. Képes lesz a program hibáinak megtalálására és az önálló javításra. A hatodik fejezet a csomagokkal és a láthatóságokkal foglalkozó feladatokat tartalmaz.A hallgató így megismeri a fejlesztői és felhasználói dokumentációk készítésének alapvető módszertani eszközeit. Képes lesz alapvető szoftverfejlesztési technológiák alkalmazására, valamint fejlesztői és felhasználói dokumentációk készítésére. Ezáltal megérti az élethosszig tartó tanulás jelentőségét, törekszik ennek megvalósítására, a folyamatos szakmai képzésre és általános önképzésre. Tudatában lesz az általa használt és működtetett informatikai eszközparkok értékének és jelentőségének, azokért személyes felelősséget vállal.A hallgató a feladatok segítségével megsmeri az azonosító, hi-vatkozási kör, takarás, alapértelmezés szerinti kezdeti érték, this objektumreferencia, konstruktor valamint az inicializálók fogalmát. A fejlesztőrendszerben objektumokat inicializál, konstruktorokat hoz létre, objektumokat a referenciájukkal kezel. Elkötelezett lesz az objektumok és referenciáik pontos használata iránt. Önállóan végez elemi objektumokkal kapcsolatos műveleteket, para-

méterátadás t és ezeket önállóan, pontosan kódolja. A hetedik fejezet feladatai az öröklődés témájához tartoznak. A hallgató ezek által tisztában lesz az öröklődés, a típuskonverziók, az utódosztály adatai és kapcsolatai, a metódus felülírása, a dinamikus és statikus kötés, a konstruktorláncolás technikájával, valamint a polimorfizmus, absztrakt metódus, absztrakt osztály, láthatóság, interfészek és belső osztályok fogalmával. Megismeri az alapvető kivételkezelés eljárásokat (továbbadás, elkapás, lekezelés), valamint az állománykezelést, a folyamokat (bájtfolyam, karakterfolyam, adatfolyam, objektumfolyam). Megismeri továbbá a legfontosabb osztályokat, interfészeket, kollekciókat. Munkája során megérti és használja az osztályok öröklтetését, a típuskonverzió technikáját. Felismeri a dinamikus és statikus kötést. Alkalmazza a konstruktorok láncolását. Igyekszik meghatározni az objektumok adatait, szem előtt tartja, hogy az öröklődés milyen esetekben alkalmazható. Önállóan meg tudja határozni az egyes objektumok konstruktorait. Felismeri a polimorfizmust, absztrakt osztályt és metódust készít, képes lesz interfészt implementálni. Törekszik a pontos kódolásra a polimorfizmus és az interfészek használata során. Figyelembe veszi a különböző lehetőségeket, elkötelezett az interfészek használata iránt.Betartja az absztrakt osztály használatának szabályait. Korrigálja saját hibáit. Képes lesz alapvető kivételkezelő eljárások elkészítésére, saját kivételek létrehozására. Jártas lesz az állománykezeléssel kapcsolatos feladatok megoldásában. Képes lesz objektumokat szerializálni. Jártas lesz a leggyakrabban alkalmazott osztályok használatában. Érdeklődik a valós élet problémái iránt, belátja az objektumorientált programozás hasznosságát. Kíváncsi lesz az állománykezeléssel kapcsolatos feladatok megoldására, nyitott a különböző megoldási módszerekre. Belátja, hogy milyen feladatoknál milyen osztályokat, kollekciókat érdemes használni. Végül önállóan készít objektumorientált programokat, képes lesz az önellenőrzésre és a hibák önálló javítására. Önállóan alkalmazza a kollekciókat az objektumorientált programokban.

A tantárggyal a következő konkrét *tanulási eredmények* alakíthatóak ki:

Tudás	**Képesség**	**Attitűd**	**Autonómia/ felelősség**
Ismeri a legfontosabb vezérlőszerkezeteket, így a szelekciókat és az iterációkat.	Szelekciós és iterációs utasításokat alkalmaz az elkészített kódban.	Törekszik a szelekciók és iterációk pontos megvalósítására.	Képes önállóan szelekciós és iterációs utasítások használatára.
Ismeri a metódusokat, a metódusok felépítését, azok lokális változóinak szerepét, valamint a rekurziót.	Rekurzív metódusokat alkalmaz.	Igyekszik a rekurzív metódusok pontos használatára.	Betartja a rekurzív metódushívás szabályait.

Tudás	**Képesség**	**Attitűd**	**Autonómia/ felelősség**
Legyen tisztában az osztályok valamint az objektumok használatával, készítésével (referenciatípus, objektum létrehozása, deklarálása)	Egyszerűbb osztályokat és objektumokat készít, objektumokat deklarál.	Törekszik az objektumok és az osztályok precíz használatára.	A példatár segítségével önállóan ellenőrzi a munkáját.
Ismeri az osztály fogalmát, felépítését, deklarációit, valamint az osztálytag és példánytag fogalmát.	Osztály- és példánytagokat tartalmazó osztályokat hoz létre.	Nyitott az objektumok különböző módszerekkel történő létrehozására, az objektumok különböző célú alkalmazására.	Önállóan megtalálja és javítja a program hibáit.
Ismeri az azonosító, hivatkozási kör, takarás, alapértelmezés szerinti kezdeti érték, this objektumreferencia, konstruktor valamint az inicializálók fogalmát.	Fejlesztőrendszerben objektumokat inicializál, konstruktorokat létrehoz, objektumokat a referenciájukkal kezel.	Elkötelezett az objektumok és referenciáik pontos használata iránt.	Önállóan végez elemi objektumokkal kapcsolatos műveleteket, paraméterátadást és ezeket önállóan, pontosan kódolja.
Tisztában van az öröklődés, a típuskonverziók, az utódosztály adatai és kapcsolatai, a metódus felülírása, a dinamikus és statikus kötés, valamint a konstruktorláncolás technikájával.	Megérti és használja az osztályok örököltetését, a típuskonverzió technikáját. Felismeri a dinamikus és statikus kötést. Alkalmazza a konstruktorok láncolását.	Törekszik az objektumok adatainak pontos meghatározására. Szem előtt tartja, hogy az öröklődés milyen esetekben alkalmazható.	Önállóan meghatározza az egyes objektumok konstruktorait.

Tudás	**Képesség**	**Attitűd**	**Autonómia/ felelősség**
Tisztában van a polimorfizmus, absztrakt metódus, absztrakt osztály, láthatóság, interfészek és belső osztályok fogalmával.	Felismeri a polimorfizmust. Absztrakt osztályt és metódust készít. Képes interfészt implementálni.	Törekszik a pontos kódolásra a polimorfizmus és az interfészek használata során. Figyelembe veszi a különböző lehetőségeket, elkötelezett az interfészek használata iránt.	Betartja az absztrakt osztály használatának szabályait. Korrigálja saját hibáit.
Ismeri az alapvető kivételkezelés eljárásokat (továbbadás, elkapás, lekezelés).	Képes alapvető kivételkezelő eljárások elkészítésére, saját kivételek létrehozására.	Érdeklődik a valós élet problémái iránt, belátja az objektumorientált programozás hasznosságát.	Önállóan készít objektumorientált programokat.
Ismeri az állománykezelést, a folyamokat (bájtfolyam, karakterfolyam, adatfolyam, objektumfolyam).	Jártas az állománykezeléssel kapcsolatos feladatok megoldásában. Objektumokat szerializál.	Kíváncsi az állománykezeléssel kapcsolatos feladatok megoldására. Nyitott a különböző megoldási módszerekre.	Önállóan ellenőrzi, és amennyiben szükséges, javítja a munkáját.
Ismeri a legfontosabb osztályokat, interfészeket (pl. java.util vagy java.io csomag elemei).	Jártas a leggyakrabban alkalmazott osztályok, kollekciók használatában (pl. java.util vagy java.io csomag osztályai és kollekciói).	Belátja, hogy milyen feladatoknál milyen osztályokat, kollekciókat érdemes használni.	Önállóan alkalmazza a kollekciókat az objektumorientált programokban.

1 Vezérlési szerkezetek

1. A program olvasson be tetszőleges egész számot parancssori argumentumból, és határozza meg azok minimumát. Szóljon, ha ehhez nincs elég (legalább 2) bemenet.

```
public class Feladat1_1{
  public static void main(String[] args){
    if(args.length<2){
      System.out.println("Nincs eleg bemenet!");
      System.exit(0);
    }
    int min=Integer.MAX_VALUE;
    for(int i=0;i<args.length;i++){
      int act=Integer.parseInt(args[i]);
      if(act<min){
        min=act;
      }
    }
    System.out.println("A legkisebb szam: "+min);
  }
}
```

Feladat_1_1 megoldása.

2. A program olvasson be tetszőleges egész számot parancssori argumentumból, és határozza meg azok maximumát. Szóljon, ha ehhez nincs elég (legalább 2) bemenet.

```
public class Feladat1_2{
  public static void main(String[] args){
    if(args.length<2){
      System.out.println("Nincs eleg bemenet!");
      System.exit(0);
    }
    int max=Integer.MIN_VALUE;
    for(int i=0;i<args.length;i++){
```

```
      int act=Integer.parseInt(args[i]);
      if(act>max){
        max=act;
      }
    }
    System.out.println("A legnagyobb szam: "+max);
  }
}
```

Feladat_1_2 megoldása.

3. A program olvasson be tetszőleges egész számot parancssori argumentumból, és határozza meg azok összegét. Szóljon, ha ehhez nincs elég (legalább 2) bemenet.

```
public class Feladat1_3{
  public static void main(String[] args){
    if(args.length<2){
      System.out.println("Nincs eleg bemenet!");
      System.exit(0);
    }
    int sum=0;
    for(int i=0;i<args.length;i++){
      int act=Integer.parseInt(args[i]);
      sum+=act;
    }
    System.out.println("A szamok osszege: "+sum);
  }
}
```

Feladat_1_3 megoldása.

4. A program olvasson be tetszőleges egész számot parancssori argumentumból, és határozza meg azok átlagát. Szóljon, ha ehhez nincs elég (legalább 2) bemenet.

```
public class Feladat1_4{
  public static void main(String[] args){
    if(args.length<2){
      System.out.println("Nincs eleg bemenet!");
      System.exit(0);
    }
    double sum=0;
    for(int i=0;i<args.length;i++){
      double act=Double.parseDouble(args[i]);
```

```
10        sum+=act;
11      }
12      double average=sum/args.length;
13      System.out.println("A szamok atlaga: "+average);
14    }
15  }
```

Feladat_1_4 megoldása.

5. A program olvasson be tetszőleges egész számot parancssori argumentumból, és határozza meg a beolvasott pozitív számok minimumát, valamint a negatív számok maximumát. Szóljon, ha ehhez nincs elég (legalább 2) bemenet.

```
1  public class Feladat1_5{
2    public static void main(String[] args){
3      if(args.length<2){
4        System.out.println("Nincs eleg bemenet!");
5        System.exit(0);
6      }
7      int posmin=Integer.MAX_VALUE;
8      int negmax=Integer.MIN_VALUE;
9      for(int i=0;i<args.length;i++){
10       int act=Integer.parseInt(args[i]);
11       if(act>=0){
12         if(act<posmin){
13           posmin=act;
14         }
15       }else{
16         if(act>negmax){
17           negmax=act;
18         }
19       }
20     }
21     System.out.println("A pozitiv szamok minimuma:"+posmin);
22     System.out.println("A negativ szamok maximuma:"+negmax);
23   }
24 }
```

Feladat_1_5 megoldása.

6. A program olvasson be tetszőleges egész számot parancssori argumentumból, és számolja meg, hogy összesen hány pozitív számot olvasott be. Szóljon, ha nem kap bemeneti paramétereket.

```java
public class Feladat1_6{
  public static void main(String[] args){
    if(args.length<1){
      System.out.println("Nincs eleg bemenet!");
      System.exit(0);
    }
    int count=0;
    for(int i=0;i<args.length;i++){
      int act=Integer.parseInt(args[i]);
      if(act>=0){
        count++;
      }
    }
    System.out.println("A pozitiv szamok darabszama:"+count);
  }
}
```

Feladat_1_6 megoldása.

7. A program olvasson be tetszőleges egész számot parancssori argumentumból, és számolja meg, hogy összesen hány negatív számot olvasott be. Szóljon, ha nem kap bemeneti paramétereket.

```java
public class Feladat1_7{
  public static void main(String[] args){
    if(args.length<1){
      System.out.println("Nincs eleg bemenet!");
      System.exit(0);
    }
    int count=0;
    for(int i=0;i<args.length;i++){
      int act=Integer.parseInt(args[i]);
      if(act<0){
        count++;
      }
    }
    System.out.println("A negativ szamok darabszama:"+count);
  }
}
```

Feladat_1_7 megoldása.

8. A program olvasson be tetszőleges egész számot parancssori argumentumból, és számolja meg, hogy összesen hány nullát olvasott be. Szóljon, ha nem kap bemeneti paramétereket.

```
public class Feladat1_8{
  public static void main(String[] args){
    if(args.length<1){
      System.out.println("Nincs eleg bemenet!");
      System.exit(0);
    }
    int count=0;
    for(int i=0;i<args.length;i++){
      int act=Integer.parseInt(args[i]);
      if(act==0){
        count++;
      }
    }
    System.out.println("A nullak darabszama:"+count);
  }
}
```

Feladat_1_8 megoldása.

9. A program olvasson be tetszőleges egész számot parancssori argumentumból, és írja ki, hogy a páratlan, vagy a páros számok összege a nagyobb.

```
public class Feladat1_9{
  public static void main(String[] args){
    if(args.length<1){
      System.out.println("Nincs eleg bemenet!");
      System.exit(0);
    }
    int evensum=0;
    int oddsum=0;
    for(int i=0;i<args.length;i++){
      int act=Integer.parseInt(args[i]);
      if(act%2==0){
        evensum+=act;
      }else{
        oddsum+=act;
      }
    }
    if(evensum>oddsum){
      System.out.println("A paros szamok osszege a nagyobb.");
```

```
    }else if(oddsum>evensum){
      System.out.println("A paratlan szamok osszege a nagyobb.");
    }else{
      System.out.println("A paros es a paratlan szamok osszege
         egyenlo.");
    }
  }
}
```

Feladat_1_9 megoldása.

10. A program olvasson be két egész számot (x és y) parancssori argumentumból. Írja ki az x és y között található összes egész számot (x -> y felé haladva), valamint szóljon, ha a bemenet nem megfelelő formátumú (itt több problémát is vizsgálni kell).

```
public class Feladat1_10{
  public static void main(String[] args){
    if(args.length!=2){
      System.out.println("Nem megfelelo szamu bemenet.");
      System.exit(0);
    }
    int x=Integer.parseInt(args[0]);
    int y=Integer.parseInt(args[1]);
    if(y<=x){
      System.out.println("A masodik szamnak nagyobbnak kell lennie
         .");
      System.exit(0);
    }
    System.out.print(x+" es "+y+" kozotti szamok: ");
    for(int i=x+1;i<y;i++){
      System.out.print(i+" ");
    }
  }
}
```

Feladat_1_10 megoldása.

11. Számtani sorozat (egész számokból) első n elemének kiszámolása tömb segítségével. A program a sorozat kezdőelemét, a differenciát, illetve n értékét parancssori argumentumból olvassa, valamint szóljon, ha nem megfelelő a bemeneti paraméterek száma.

```
public class Feladat1_11{
  public static void main(String[] args){
    if(args.length != 3){
```

```
      System.out.println("Nem megfelelo szamu bemenet.");
      System.exit(0);
    }
    int first=Integer.parseInt(args[0]);
    int diff=Integer.parseInt(args[1]);
    int n=Integer.parseInt(args[2]);
    int[] progression = new int[n];
    progression[0]=first;
    for(int i=1;i<progression.length;i++){
      progression[i]=progression[i-1]+diff;
    }
    System.out.print("A sorozat elemei: ");
    for(int i=0;i<progression.length;i++){
      System.out.print(progression[i]+" ");
    }
  }
}
```

Feladat_1_11 megoldása.

12. Mértani sorozat (egész számokból) első n elemének kiszámolása tömb segítségével. A program a sorozat kezdőelemét, a kvócienst, illetve n értékét parancssori argumentumból olvassa, valamint szóljon, ha nem megfelelő a bemeneti paraméterek száma.

```
public class Feladat1_12{
  public static void main(String[] args){
    if(args.length != 3){
      System.out.println("Nem megfelelo szamu bemenet.");
      System.exit(0);
    }
    int first=Integer.parseInt(args[0]);
    int r=Integer.parseInt(args[1]);
    int n=Integer.parseInt(args[2]);
    int[] progression = new int[n];
    progression[0]=first;
    for(int i=1;i<progression.length;i++){
      progression[i]=progression[i-1]*r;
    }
    System.out.print("A sorozat elemei: ");
    for(int i=0;i<progression.length;i++){
      System.out.print(progression[i]+" ");
    }
```

```
19    }
20  }
```

Feladat_1_12 megoldása.

13. Fibonacci sorozat első n elemének kiszámítása tömb segítségével. n értékét a program parancssori argumentumként kérje, és szóljon, ha nem pontosan egy paramétert kapott.

```
1  public class Feladat1_13{
2    public static void main(String[] args){
3      if(args.length!=1){
4        System.out.println("Nem megfelelo szamu parameter!");
5        System.exit(0);
6      }
7      int n=Integer.parseInt(args[0]);
8      int[] fibo=new int[n];
9      fibo[0]=1;
10     fibo[1]=1;
11     for(int i=2;i<fibo.length;i++){
12       fibo[i]=fibo[i-1]+fibo[i-2];
13     }
14     System.out.print("A Fibonacci sorozat elemei: ");
15     for(int i=0;i<fibo.length;i++){
16       System.out.print(fibo[i]+" ");
17     }
18   }
19 }
```

Feladat_1_13 megoldása.

14. Tetszőleges számú szó beolvasása parancssori argumentumból. A program számolja meg, hogy milyen hosszúságú szóból mennyi van, ezeket az értékeket tárolja tömbben (pl. 1 hosszú szó 3 db, 2 hosszú szó 0 db, 3 hosszú szó 4 db, stb.). A kapott értékeket írja ki, valamint szóljon, ha nincs megadva bemeneti szó.

```
1  public class Feladat1_14{
2    public static void main(String[] args){
3      if(args.length<1){
4        System.out.println("Nem megfelelo szamu parameter!");
5        System.exit(0);
6      }
7      int maxlength=0;
8      for(int i=0;i<args.length;i++){
9        if(args[i].length()>maxlength){
```

```
10          maxlength=args[i].length();
11        }
12      }
13      int[] sizedist=new int[maxlength+1];
14      for(int i=0;i<args.length;i++){
15        sizedist[args[i].length()]++;
16      }
17      for(int i=1;i<sizedist.length;i++){
18        System.out.println(i+" hosszu szavak szama: "+ sizedist[i]);
19      }
20    }
21  }
```

Feladat_1_14 megoldása.

15. A program olvasson be 3 számot parancssori argumentumból. Szóljon, ha nincs elég bemene. Döntse el, hogy a számok lehetnek -e egy háromszög oldalhosszai. Ha igen, számolja ki annak keürletét és területét. A terület kiszámítására alkalmazza Heron képletét: $T = \sqrt{(s*(s-a)*(s-b)*(s-c))}$, ahol s a kerület fele.

```
1  public class Feladat1_15{
2    public static void main(String[] args){
3      if(args.length!=3){
4        System.out.println("Nem megfelelo szamu parameter!");
5        System.exit(0);
6      }
7      double a=Double.parseDouble(args[0]);
8      double b=Double.parseDouble(args[1]);
9      double c=Double.parseDouble(args[2]);
10     double s= (a+b+c)/2;
11     if(a+b>c && a+c>b && b+c>a){
12       double area=Math.sqrt(s*(s-a)*(s-b)*(s-c));
13       System.out.println("A haromszog terulete: "+ area);
14     }else{
15       System.out.println("A kapott szamok nem lehetnek egy
             haromszog oldalai.");
16     }
17   }
18 }
```

Feladat_1_15 megoldása.

16. A program olvasson be egy n egész számot parancssori argumentumból. Szóljon, ha nincs bemenet. Dobjon egy hatoldalú kockával n alkalommal (véletlenszerű értékeket), és ment-

se tömbbe, hogy az egyes értékeket hányszor dobta ki. Ezután írja ki, hogy az különböző értékek az dobások hány százalékában fordultak elő.

```
public class Feladat1_16{
  public static void main(String[] args){
    if(args.length!=1){
      System.out.println("Nem megfelelo szamu parameter!");
      System.exit(0);
    }
    int n=Integer.parseInt(args[0]);
    int[] results = new int[6];
    for(int i=0;i<n;i++){
      int random=(int)(Math.random()*6+1);
      results[random-1]++;
    }
    for(int i=0;i<results.length;i++){
      System.out.println((results[i]/(double)n*100)+"%");
    }
  }
}
```

Feladat_1_16 megoldása.

17. A program olvasson be legalább 2 számot parancssori argumentumból, és szóljon, ha nincs elég bemenet. Vizsgálja meg, hogy a kapott számok sorrendje növekvő, csökkenő, vagy egyik sem, és ezt írja ki.

```
public class Feladat1_17{
  public static void main(String[] args){
    if(args.length<2){
      System.out.println("Nem megfelelo szamu parameter!");
      System.exit(0);
    }
    int counter=0;
    for(int i=1;i<args.length;i++){
      if(Integer.parseInt(args[i-1])<Integer.parseInt(args[i])){
        counter++;
      }else if (Integer.parseInt(args[i-1])>Integer.parseInt(args[
         i])){
        counter--;
      }
    }
    if(counter==(args.length-1)){
```

```
16        System.out.println("A sorozat novekvo");
17      }else if(counter == -(args.length-1)){
18        System.out.println("A sorozat csokkeno");
19      }else{
20        System.out.println("Egyik sem");
21      }
22    }
23  }
```

Feladat_1_17 megoldása.

18. Beérkező rendelések összértékét szeretnénk meghatározni. Ehhez parancssorról a rendelések sorszámait kapjuk meg, ami tetszőleges darab egész szám formájában jön. Négy különböző rendelésünk van, az 1-es 1000 Ft, 2-es 5000 Ft, 3-as 7000 Ft és a 4-es 9000 Ft. Egy rendelés sorszáma többször is szerepelhet. Adjuk össze az érkező sorszámoknak megfelelő értékeket, és az összeget írjuk ki. Ha olyan szám szerepel, ami nem rendes sorszám (nem 1,2,3,4), akkor minden esetben írjuk ki konzolra, hogy "Nem megfelelo sorszam". A program szóljon, ha nincs legalább 1 bemenet.

```
1  public class Feladat1_18{
2    public static void main(String[] args){
3      if(args.length<1){
4        System.out.println("Nem megfelelo szamu parameter!");
5        System.exit(0);
6      }
7      int sum=0;
8      for(int i=0;i<args.length;i++){
9        int actual=Integer.parseInt(args[i]);
10       switch(actual){
11         case 1: sum+=1000; break;
12         case 2: sum+=5000; break;
13         case 3: sum+=7000; break;
14         case 4: sum+=9000; break;
15         default: System.out.println("Nem megfelelo sorszam");
16       }
17     }
18     System.out.println("A rendelesek osszerteke: "+ sum);
19   }
20 }
```

Feladat_1_18 megoldása.

19. Készítsünk programot, ami kiszámolja egy szabályos sokszög kerületét. A program két bemenetet vár parancssori argumentumból: a sokszög oldalainak a számát, valamint az

oldalhosszat. A probléma csak az, hogy a programozási nyelvünkben "elromlott" a szorzás, így ezt a műveletet nem használhatjuk a feladat megoldása során. A program szóljon, ha nem 2 bemenetet kapott.

```
public class Feladat1_19{
  public static void main(String[] args){
    if(args.length!=2){
      System.out.println("Nem megfelelo szamu parameter!");
      System.exit(0);
    }
    int n=Integer.parseInt(args[0]);
    int length=Integer.parseInt(args[1]);
    int result=0;
    for(int i=0;i<n;i++){
      result=result+length;
    }
    System.out.println("A sokszog kerulete:"+result);
  }
}
```

Feladat_1_19 megoldása.

20. Egy könyv olvasása közben kíváncsiak leszünk, hogy melyik lehet a benne található leghosszabb szó, ezért készítünk egy programot, ami eldönti ezt nekünk. A program a könyvben található szavakat kapja bemenetnek parancssori argumentumból (tetszőleges mennyiségű szó). Írjuk ki, hogy melyik a bemenetben található leghosszabb szó (ha több leghosszabbat is találunk, akkor bármelyiket kiírhatjuk). A program szóljon, ha nem kapott bemenetet.

```
public class Feladat1_20{
  public static void main(String[] args){
    if(args.length<1){
      System.out.println("Nem megfelelo szamu parameter!");
      System.exit(0);
    }
    int maxlength=0;
    String maxstring = new String();
    for(int i=0;i<args.length;i++){
      if(args[i].length()>maxlength){
        maxstring=args[i];
        maxlength=args[i].length();
      }
    }
```

```
15      System.out.println("A leghosszabb szo:"+maxstring);
16    }
17  }
```

Feladat_1_20 megoldása.

21. Matek ZH-ra készülés közben elromlott a számológépünk, ezért elhatározzuk, hogy írunk egy programot, ami helyettesíteni fogja. A számológépünk mindig két értéken végez el műveletet, ezért a program pontosan 3 bemenetet kapjon parancssori argumentumból: az első számot, az operátort, valamin a második számot (pl. 3 + 1). Írjuk ki, hogy mi lesz a megadott kifejezés eredménye. A program szóljon, ha nem 3 bemenetet kapott. (tipp: szorzásnál * helyett más karaktert használjunk bemenetnek, pl. x)

```
1  public class Feladat1_21{
2    public static void main(String[] args){
3      if(args.length!=3){
4        System.out.println("Nem megfelelo szamu parameter!");
5        System.exit(0);
6      }
7      double first=Double.parseDouble(args[0]);
8      String operator=args[1];
9      double second=Double.parseDouble(args[2]);
10     double result=0;
11     switch(operator){
12       case "+": result=first+second; break;
13       case "-": result=first-second; break;
14       case "/": result=first/second; break;
15       case "x": result=first*second; break;
16       default: System.out.println("Nincs ilyen muvelet");
17     }
18     System.out.println("Az eredmeny: "+result);
19   }
20 }
```

Feladat_1_21 megoldása.

22. Az előző feladatban elkészített számológépünk nagyon jól működik, de rövidesen rájövünk, hogy hosszabb kifejezéseket is ki szeretnénk számolni. Írunk tehát egy új programot, ami az alábbi felépítésű bemenetet kapja: műveletek száma (n), kezdőérték, majd n*{operátor, operátorhoz tartozó érték}. (pl. 3 1 + 2 x 2 - 1 → 3 művelet, 1 a kezdőérték, és először +2-t végezzük el, majd *2-t, majd -1-t). A program számolja ki az így kapott műveletsorrend eredményét. A műveleteket abban a sorrendben hajtsa végre, ahogy a bemenetben szerepelnek. A program szóljon, ha nem megfelelő a bemenet.

```java
public class Feladat1_22{
  public static void main(String[] args){
    if(args.length<3){
      System.out.println("Nem megfelelo szamu parameter!");
      System.exit(0);
    }
    int n=Integer.parseInt(args[0]);
    if(args.length!=n*2+2){
      System.out.println("Nem megfelelo formatumu bemenet!");
      System.exit(0);
    }
    double result=Double.parseDouble(args[1]);
    for(int i=2;i<2*n+1;i+=2){
      String operator=args[i];
      double number=Integer.parseInt(args[i+1]);
      switch(operator){
      case "+": result=result+number; break;
      case "-": result=result-number; break;
      case "/": result=result/number; break;
      case "x": result=result*number; break;
      default: System.out.println("Nincs ilyen muvelet");
      }
    }
    System.out.println("Az eredmeny:"+result);
  }
}
```

Feladat_1_22 megoldása.

23. Jancsi és Juliska az erdőben sétálva minden kereszteződésnél morzsákat szór el. Jancsi az első kereszteződésben 3 szemet, Juliska a második kereszteződésben 2 szemet. Minden további kereszteződésben annyi szemet szórnak el, mint az előző kettőben összesen. A sétájuk az n-edik kereszteződésben ér véget, ugyanis itt egy életnagyságú mézeskalács házzal találják szemben magukat, de mielőtt bemennek, még itt is elszórnak valamennyi morzsát. Mennyi morzsát szórtak el a ház előtt? A kereszteződések számát (n értékét) parancssori argumentumból kapjuk bemenetnek. A program írja ki az utolsó kereszteződésben elszórt morzsa mennyiségét. A program szóljon, ha nem kapott bemenetet.

```java
public class Feladat1_23{
  public static void main(String[] args){
    if(args.length<1){
      System.out.println("Nem megfelelo szamu parameter!");
```

```
      System.exit(0);
    }
    int n=Integer.parseInt(args[0]);
    int[] array=new int[n];
    array[0]=3;
    array[1]=2;
    for(int i=2;i<array.length;i++){
      array[i]=array[i-1]+array[i-2];
    }
    System.out.println("Az utolso keresztezodesben levo morzsak
       szama:"+array[n-1]);
  }
}
```

Feladat_1_23 megoldása.

24. Kisboltunkba egy szállítmány dinnye érkezik, amit átvétel előtt leellenőrzünk, lemérjük mindegyiket. A dinnyék súlyai lesznek a program bemenete, ezeket parancssori argumentumból olvassuk be (tetszőleges darab egész szám). Tudjuk, hogy ha egy dinnye súlya páros szám, és osztható 3-al is, akkor biztos rohadt lesz, ezeket nem szeretnénk átvenni. A program állapítsa meg a súlyok alapján, hogy hány problémás dinnye található a szállítmányban. A program szóljon, ha nem kapott bemenetet.

```
public class Feladat1_24{
  public static void main(String[] args){
    if(args.length<1){
      System.out.println("Nem megfelelo szamu parameter!");
      System.exit(0);
    }
    int count=0;
    for(int i=0;i<args.length;i++){
      int actual=Integer.parseInt(args[i]);
      if(actual%2==0 && actual%3==0){
        count++;
      }
    }
    System.out.println("A rohadt dinnyek szama:"+count);
  }
}
```

Feladat_1_24 megoldása.

25. Készítsünk programot, ami megadott háromszögek közül kiválasztja, hogy melyiknek a legnagyobb a kerülete. A program parancssori argumentumból kapja a bemenetét az

alábbi módon: először a háromszögek darabszáma (n), utána hármasával az oldalhosszak (n*{oldal1 oldal2 oldal3}) következnek (pl. 2 1 2 3 3 3 4 → 2 háromszög, az egyik oldalai 1,2,3, a másik oldalai 3,3,4). Állapítsuk meg, hogy hanyadik háromszögnek a legnagyobb a kerülete. A program szóljon, ha nem megfelelő a bemenet.

```
public class Feladat1_25{
  public static void main(String[] args){
    if(args.length<3){
      System.out.println("Nem megfelelo szamu parameter!");
      System.exit(0);
    }
    int n=Integer.parseInt(args[0]);
    if(args.length!=n*3+1){
      System.out.println("Nem megfelelo formatumu bemenet!");
      System.exit(0);
    }
    int counter=1;
    int maxitem=1;
    int max=0;
    for(int i=1;i<n*3+1;i+=3){
      int act=Integer.parseInt(args[i])+Integer.parseInt(args[i
         +1])+Integer.parseInt(args[i+2]);
      if(act>max){
        maxitem=counter;
        max=act;
      }
      counter++;
    }
    System.out.println("A legnagyobb haromszog sorszama:"+maxitem)
       ;
  }
}
```

Feladat_1_25 megoldása.

26. Reklámozni szeretnénk a legújabb termékünket, és készítünk egy programot, ami elektronikus hirdetőtáblaként megteszi ezt nekünk. A hirdetőtábla egymás után szövegeket fog kiírni a képernyőre: vagy a termék nevét, vagy a termék árát. A program 3 bemenetet kap parancssori argumentumból: az első a termék neve, a második a termék ára, a harmadik pedig egy szám, hogy összesen hány sort szeretnénk kiírni a táblára. Alapesetben a hirdetőtábla mindig a termék nevét írja ki, de minden 5. kiírás a termék árát fogja kiírni ehelyett. A program szóljon, ha nem 3 bemenetet kap.

```java
public class Feladat1_26{
  public static void main(String[] args){
    if(args.length!=3){
      System.out.println("Nem megfelelo szamu parameter!");
      System.exit(0);
    }
    String name=args[0];
    int price=Integer.parseInt(args[1]);
    int n=Integer.parseInt(args[2]);
    for(int i=1;i<=n;i++){
      if(i%5==0){
        System.out.println(price);
      }else{
        System.out.println(name);
      }
    }
  }
}
```

Feladat_1_26 megoldása.

2 Metódusok

1. Készíts programot, amely a parancssori argumentumból tetszőleges darab egész számot olvas be. Szóljon, ha nincs legalább 1 bemenet, és lépjen ki. A programnak legyen egy metódusa, ami egy int tömböt vár paraméternek, és visszaadja a tömbben található számok összegét. Ezt a visszaadott értéket írja ki.

```
1  public class Feladat2_1{
2    public static void main(String[] args){
3      if(args.length<1){
4        System.out.println("Nem megfelelo szamu parameter!");
5        System.exit(0);
6      }
7      int[] array=new int[args.length];
8      for(int i=0;i<args.length;i++){
9        array[i]=Integer.parseInt(args[i]);
10     }
11     int result=sum(array);
12     System.out.println("A tombben talalhato szamok osszege:"+
          result);
13   }
14   public static int sum(int[] array){
15     int result=0;
16     for(int i=0;i<array.length;i++){
17       result+=array[i];
18     }
19     return result;
20   }
21 }
```

Feladat_2_1 megoldása.

2. Készíts programot, amely a parancssori argumentumból tetszőleges darab egész számot

olvas be. Szóljon, ha nincs legalább 1 bemenet, és lépjen ki. A programnak legyen egy metódusa, ami egy int tömböt vár paraméternek, és visszaadja a tömbben található számok átlagát. Ezt a visszaadott értéket írja ki.

```
public class Feladat2_2{
  public static void main(String[] args){
    if(args.length<1){
      System.out.println("Nem megfelelo szamu parameter!");
      System.exit(0);
    }
    int[] array=new int[args.length];
    for(int i=0;i<args.length;i++){
      array[i]=Integer.parseInt(args[i]);
    }
    double result=mean(array);
    System.out.println("A tombben talalhato szamok atlaga:"+result
       );
  }
  public static double mean(int[] array){
    double result=0;
    for(int i=0;i<array.length;i++){
      result+=array[i];
    }
    return result/array.length;
  }
}
```

Feladat_2_2 megoldása.

3. Készíts programot, amely a parancssori argumentumból tetszőleges darab egész számot olvas be. Szóljon, ha nincs legalább 1 bemenet, és lépjen ki. A programnak legyen egy metódusa, ami egy int tömböt vár paraméternek, és visszaadja a tömbben található legkisebb elem értékét. Ezt a visszaadott értéket írja ki.

```
public class Feladat2_3{
  public static void main(String[] args){
    if(args.length<1){
      System.out.println("Nem megfelelo szamu parameter!");
      System.exit(0);
    }
    int[] array=new int[args.length];
    for(int i=0;i<args.length;i++){
      array[i]=Integer.parseInt(args[i]);
```

```
10      }
11      int result=min(array);
12      System.out.println("A tombben talalhato legkisebb elem:"+
           result);
13    }
14    public static int min(int[] array){
15      int min=Integer.MAX_VALUE;
16      for(int i=0;i<array.length;i++){
17        if(array[i]<min){
18          min=array[i];
19        }
20      }
21      return min;
22    }
23  }
```

Feladat_2_3 megoldása.

4. Készíts programot, amely a parancssori argumentumból tetszőleges darab egész számot olvas be. Szóljon, ha nincs legalább 1 bemenet, és lépjen ki. A programnak legyen egy metódusa, ami egy int tömböt vár paraméternek, és visszaadja a tömbben található legnagyobb elem értékét. Ezt a visszaadott értéket írja ki.

```
1   public class Feladat2_4{
2     public static void main(String[] args){
3       if(args.length<1){
4         System.out.println("Nem megfelelo szamu parameter!");
5         System.exit(0);
6       }
7       int[] array=new int[args.length];
8       for(int i=0;i<args.length;i++){
9         array[i]=Integer.parseInt(args[i]);
10      }
11      int result=max(array);
12      System.out.println("A tombben talalhato legnagyobb elem:"+
           result);
13    }
14    public static int max(int[] array){
15      int max=Integer.MIN_VALUE;
16      for(int i=0;i<array.length;i++){
17        if(array[i]>max){
18          max=array[i];
19        }
```

```
20      }
21      return max;
22    }
23  }
```

Feladat_2_4 megoldása.

5. Készíts programot, amely a parancssori argumentumból tetszőleges darab egész számot olvas be. Szóljon, ha nincs legalább 1 bemenet, és lépjen ki. A programnak legyen két metódusa: mindkettő egy-egy int tömböt vár paraméternek, és visszaadja a tömbben található legkisebb, illetve legnagyobb elem értékét. A visszaadott értékekről döntse el, hogy a legnagyobb elem osztható -e a legkisebb elemmel, és írjon ki igen/nem szöveget ettől függően.

```
1  public class Feladat2_5{
2    public static void main(String[] args){
3      if(args.length<1){
4        System.out.println("Nem megfelelo szamu parameter!");
5        System.exit(0);
6      }
7      int[] array=new int[args.length];
8      for(int i=0;i<args.length;i++){
9        array[i]=Integer.parseInt(args[i]);
10     }
11     int max=max(array);
12     int min=min(array);
13     if(max%min==0){
14       System.out.println("igen");
15     }else{
16       System.out.println("nem");
17     }
18   }
19   public static int max(int[] array){
20     int max=Integer.MIN_VALUE;
21     for(int i=0;i<array.length;i++){
22       if(array[i]>max){
23         max=array[i];
24       }
25     }
26     return max;
27   }
28   public static int min(int[] array){
29     int min=Integer.MAX_VALUE;
```

```
    for(int i=0;i<array.length;i++){
      if(array[i]<min){
        min=array[i];
      }
    }
    return min;
  }
}
```

Feladat_2_5 megoldása.

6. Készíts programot, amely a parancssori argumentumból tetszőleges darab egész számot olvas be. Szóljon, ha nincs legalább 1 bemenet, és lépjen ki. A programnak legyen egy metódusa, ami egy int tömböt vár paraméternek, és visszaadja, hogy hány páros szám található a tömbben. Ezt a visszaadott értéket írja ki.

```
public class Feladat2_6{
  public static void main(String[] args){
    if(args.length<1){
      System.out.println("Nem megfelelo szamu parameter!");
      System.exit(0);
    }
    int[] array=new int[args.length];
    for(int i=0;i<args.length;i++){
      array[i]=Integer.parseInt(args[i]);
    }
    int count=even(array);
    System.out.println("Ennyi paros szam van a tombben:"+count);
  }
  public static int even(int[] array){
    int count=0;
    for(int i=0;i<array.length;i++){
      if(array[i]%2==0){
        count++;
      }
    }
    return count;
  }
}
```

Feladat_2_6 megoldása.

7. Készíts programot, amely a parancssori argumentumból tetszőleges darab egész számot olvas be. Szóljon, ha nincs legalább 1 bemenet, és lépjen ki. A programnak legyen egy

metódusa, ami egy int tömböt vár paraméternek, és visszaadja, hogy hány páratlan szám található a tömbben. Ezt a visszaadott értéket írja ki.

```
public class Feladat2_7{
  public static void main(String[] args){
    if(args.length<1){
      System.out.println("Nem megfelelo szamu parameter!");
      System.exit(0);
    }
    int[] array=new int[args.length];
    for(int i=0;i<args.length;i++){
      array[i]=Integer.parseInt(args[i]);
    }
    int count=odd(array);
    System.out.println("Ennyi paratlan szam van a tombben:"+count)
       ;
  }
  public static int odd(int[] array){
    int count=0;
    for(int i=0;i<array.length;i++){
      if(array[i]%2==1){
        count++;
      }
    }
    return count;
  }
}
```

Feladat_2_7 megoldása.

8. Készíts programot, amely a parancssori argumentumból tetszőleges darab egész számot olvas be. Szóljon, ha nincs legalább 1 bemenet, és lépjen ki. A programnak legyen egy metódusa, ami két paramétert vár: egy int tömböt, valamint egy logikai értéket, hogy a páros, vagy páratlan számokat akarjuk megszámolni a tömbben. A metódus adja vissza, hogy a logikai paraméter által meghatározott típusú számokból hány darab található a tömbben. A metódust kétszer hívd meg: számolja össze a páros és páratlan értékeket is. A program írja ki, hogy páros, vagy páratlan számokból volt több a bemenetben.

```
public class Feladat2_8{
  public static void main(String[] args){
    if(args.length<1){
      System.out.println("Nem megfelelo szamu parameter!");
      System.exit(0);
```

```
    }
    int[] array=new int[args.length];
    for(int i=0;i<args.length;i++){
      array[i]=Integer.parseInt(args[i]);
    }
    int even=evenorodd(array,true);
    int odd=evenorodd(array,false);
    if(even>odd){
      System.out.println("Paros szamokbol van tobb");
    }else if(even<odd){
      System.out.println("Paratlan szamokbol van tobb");
    }else{
      System.out.println("Ugyanannyi paros es paratlan szam van");
    }
  }
  public static int evenorodd(int[] array, boolean parity){
    int count=0;
    for(int i=0;i<array.length;i++){
      if(parity){
        if(array[i]%2==0){
          count++;
        }
      }else{
        if(array[i]%2==1){
          count++;
        }
      }
    }
    return count;
  }
}
```

Feladat_2_8 megoldása.

9. Készíts programot, amely a parancssori argumentumból tetszőleges darab egész számot olvas be. Szóljon, ha nincs legalább 2 bemenet, és lépjen ki. Az első bemenet egy lépésköz értéke lesz, minden további bemenet kerüljön egy int tömbbe. A programnak legyen egy metódusa, ami egy int tömböt, valamint egy egész lépésközt vár paraméternek, és összeadja tömb azon elemeit, amit a lépésköz határoz meg (az első elemmel kezdve). Hívja meg ezt a metódust a beolvasott értékkel és tömbbel, a visszaadott eredményt írja ki.

```
public class Feladat2_9{
  public static void main(String[] args){
```

```
    if(args.length<2){
      System.out.println("Nem megfelelo szamu parameter!");
      System.exit(0);
    }
    int n=Integer.parseInt(args[0]);
    int[] array=new int[args.length-1];
    for(int i=1;i<args.length;i++){
      array[i-1]=Integer.parseInt(args[i]);
    }
    int result=add(array,n);
    System.out.println("A kivalasztott elemek osszege:"+result);
  }
  public static int add(int[] array, int n){
    int sum=0;
    for(int i=0;i<array.length;i+=n){
      sum+=array[i];
    }
    return sum;
  }
}
```

Feladat_2_9 megoldása.

10. Készíts programot, amely a parancssori argumentumból tetszőleges darab egész számot olvas be. Szóljon, ha nincs legalább 2 bemenet, és lépjen ki. Az első bemenet egy lépésköz értéke lesz, minden további bemenet kerüljön egy int tömbbe. A programnak legyen egy metódusa, ami egy int tömböt, valamint egy egész lépésközt vár paraméternek, és összeadja tömb azon elemeit, amit a lépésköz határoz meg, és párosak (az első elemmel kezdve). Hívja meg ezt a metódust a beolvasott értékkel és tömbbel, a visszaadott eredményt írja ki.

```
public class Feladat2_10{
  public static void main(String[] args){
    if(args.length<2){
      System.out.println("Nem megfelelo szamu parameter!");
      System.exit(0);
    }
    int n=Integer.parseInt(args[0]);
    int[] array=new int[args.length-1];
    for(int i=1;i<args.length;i++){
      array[i-1]=Integer.parseInt(args[i]);
    }
    int result=add(array,n);
```

```
13      System.out.println("A kivalasztott elemek osszege:"+result);
14    }
15    public static int add(int[] array, int n){
16      int sum=0;
17      for(int i=0;i<array.length;i+=n){
18        if(array[i]%2==0){
19          sum+=array[i];
20        }
21      }
22      return sum;
23    }
24  }
```

Feladat_2_10 megoldása.

11. Készíts programot, amely a parancssori argumentumból tetszőleges darab egész számot olvas be. Szóljon, ha nincs legalább 2 bemenet, és lépjen ki. Az első bemenet egy lépésköz értéke lesz, minden további bemenet kerüljön egy int tömbbe. A programnak legyen egy metódusa 3 paraméterrel: egy int tömb, egy egész lépésköz, valamint egy logikai változó, hogy balról, vagy jobbról járja be a tömböt. A metódus adja össze a tömb azon elemeit, amit a lépésköz határoz meg (a kezdő elem a logikai értéktől függően a legelső/legutolsó legyen). Kétszer hívja meg ezt a metódust a beolvasott értékkel és tömbbel: balról és jobbról is határozza meg az összeget. Döntsd el, hogy melyik irányból kapott nagyobb értéket, és ezt írja ki.

```
1   public class Feladat2_11{
2     public static void main(String[] args){
3       if(args.length<2){
4         System.out.println("Nem megfelelo szamu parameter!");
5         System.exit(0);
6       }
7       int n=Integer.parseInt(args[0]);
8       int[] array=new int[args.length-1];
9       for(int i=1;i<args.length;i++){
10        array[i-1]=Integer.parseInt(args[i]);
11      }
12      int left=add(array,n,true);
13      int right=add(array,n,false);
14      if(left>right){
15        System.out.println("Balrol osszeadva nagyobb");
16      }else if(left<right){
17        System.out.println("Jobbrol osszeadva nagyobb");
18      }else{
```

```
19        System.out.println("Mindket oldalrol egyenlo");
20      }
21    }
22    public static int add(int[] array, int n, boolean left){
23      int sum=0;
24      if(left){
25        for(int i=0;i<array.length;i+=n){
26            sum+=array[i];
27        }
28      }else{
29        for(int i=array.length-1;i>=0;i-=n){
30            sum+=array[i];
31        }
32      }
33      return sum;
34    }
35  }
```

Feladat_2_11 megoldása.

12. Készíts programot, amely a parancssori argumentumból tetszőleges darab egész számot olvas be. Szóljon, ha nincs legalább 3 bemenet, és lépjen ki. Az első és a második bemenet egy-egy index érték lesz, minden további bemenet kerüljön egy int tömbbe. A programnak legyen egy metódusa, ami egy int tömb valahanyadik elemét határozza meg. Ehhez 3 paramétert kérjen: az tömböt, a keresendő elem indexét, valamint egy logikai értéket, hogy balról, vagy jobbról keressen a tömbben. A program hívja meg ezt a metódust kétszer: az első beolvasott indexel balról keress a tömbben, a másodikkal pedig jobbról. Mindkét eredményt írd ki.

```
1   public class Feladat2_12{
2     public static void main(String[] args){
3       if(args.length<3){
4         System.out.println("Nem megfelelo szamu parameter!");
5         System.exit(0);
6       }
7       int leftindex=Integer.parseInt(args[0]);
8       int rightindex=Integer.parseInt(args[1]);
9       int[] array=new int[args.length-2];
10      for(int i=2;i<args.length;i++){
11        array[i-2]=Integer.parseInt(args[i]);
12      }
13      int left=index(array, leftindex, true);
14      int right=index(array,rightindex, false);
```

```
15      System.out.println("Balrol keresve az eredmeny:"+left);
16      System.out.println("Jobbrol keresve az eredmeny:"+right);
17    }
18    public static int index(int[] array, int index, boolean left){
19      if(left){
20        return array[index-1];
21      }else{
22        return array[array.length-index];
23      }
24    }
25  }
```

Feladat_2_12 megoldása.

3 Rekurzív metódusok

FONTOS: Az alábbi feladatok esetében sehol nem használhatsz ciklusokat (while, for)! A feladatokat minden esetben rekurzív elven kell megoldani úgy, hogy a metódus más paraméterezéssel újra meghívja saját magát.

Írj programot, ami parancssori argumentumból egy egész számot olvas be. Szóljon, és lépjen ki, ha nincs bemenet. A programnak legyen egy metódusa, ami egy int paramétert vár. A metódus ne adjon vissza semmit, de rekurzívan írja ki az összes számot 1 és a kapott paraméter között növekvő sorrendben.

1.
```
public class Feladat2_13{
  public static void main(String[] args){
    if(args.length<1){
      System.out.println("Nem megfelelo szamu parameter!");
      System.exit(0);
    }
    int number=Integer.parseInt(args[0]);
    counter(number);
  }
  public static void counter(int n){
    if(n>1){
      counter(n-1);
      System.out.println(n);
    }else{
      System.out.println(n);
    }
  }
}
```

Feladat_2_13 megoldása.

2. Írj programot, ami parancssori argumentumból egy egész számot olvas be. Szóljon, és lépjen ki, ha nincs bemenet. A programnak legyen egy metódusa, ami egy int paramétert vár. A metódus ne adjon vissza semmit, de rekurzívan írja ki az összes számot a kapott paraméter és 1 között csökkenő sorrendben.

```java
public class Feladat2_14{
  public static void main(String[] args){
    if(args.length<1){
      System.out.println("Nem megfelelo szamu parameter!");
      System.exit(0);
    }
    int number=Integer.parseInt(args[0]);
    counter(number);
  }
  public static void counter(int n){
    if(n>1){
      System.out.println(n);
      counter(n-1);
    }else{
      System.out.println(n);
    }
  }
}
```

Feladat_2_14 megoldása.

3. Írj programot, ami parancssori argumentumból egy egész számot olvas be. Szóljon, és lépjen ki, ha nincs bemenet. A programnak legyen egy metódusa, ami egy int paramétert vár. A metódus rekurzívan adja össze az összes számot 1 és a kapott paraméter között, és adja vissza az eredményt. A metódus által visszaadott értéket írd ki.

```java
public class Feladat2_15{
  public static void main(String[] args){
    if(args.length<1){
      System.out.println("Nem megfelelo szamu parameter!");
      System.exit(0);
    }
    int number=Integer.parseInt(args[0]);
    int result=sum(number);
    System.out.println("Az eredmeny:"+result);
  }
  public static int sum(int n){
    if(n==0){
      return 0;
    }else{
      return n+sum(n-1);
    }
```

```
17    }
18  }
```

Feladat_2_15 megoldása.

4. Írj programot, ami parancssori argumentumból egy egész számot olvas be. Szóljon, és lépjen ki, ha nincs bemenet. A programnak legyen egy metódusa, ami egy int paramétert vár. A metódus rekurzívan számolja ki a kapott paraméter faktoriálisát, és adja vissza az eredményt. A metódus által visszaadott értéket írd ki.

```
1  public class Feladat2_16{
2    public static void main(String[] args){
3      if(args.length<1){
4        System.out.println("Nem megfelelo szamu parameter!");
5        System.exit(0);
6      }
7      int number=Integer.parseInt(args[0]);
8      int result=factorial(number);
9      System.out.println("Az eredmeny:"+result);
10   }
11   public static int factorial(int n){
12     if(n==1){
13       return 1;
14     }else{
15       return n*factorial(n-1);
16     }
17   }
18 }
```

Feladat_2_16 megoldása.

5. Írj programot, ami parancssori argumentumból két egész számot olvas be. Szóljon, és lépjen ki, ha nincs 2 bemenet. Az első szám egy hatványozandó érték (alap), a második pedig a hatványkitevő. A programnak legyen egy metódusa, ami két int paramétert vár: alapot és kitevőt. A metódus rekurzívan számolja ki a kapott paraméterek alapján a hatványt, és adja vissza az eredményt. A metódus által visszaadott értéket írd ki.

```
1  public class Feladat2_17{
2    public static void main(String[] args){
3      if(args.length<2){
4        System.out.println("Nem megfelelo szamu parameter!");
5        System.exit(0);
6      }
7      int base=Integer.parseInt(args[0]);
```

```
8       int exponent=Integer.parseInt(args[1]);
9       int result=power(base,exponent);
10      System.out.println("Az eredmeny:"+result);
11    }
12    public static int power(int base, int exponent){
13      if(exponent==0){
14        return 1;
15      }else if(exponent==1){
16        return base;
17      }else{
18        return base*power(base, exponent-1);
19      }
20    }
21  }
```

Feladat_2_17 megoldása.

4 Egyszerű objektumok

1. Írj osztályt, ami egy *Macska* objektumot valósít meg.
   - A macska adattagjai a következőek legyenek: név (String), súly (double), éhes -e (boolean).
   - Két konstruktort is készíts az osztályhoz. Az egyik általános legyen, ami minden adattagot a konstruktor paraméterlistájából állít be, illetve egy másik, ami az első két adattagot a konstruktor paraméterlistájából kapja, és alapértelmezetten éhes a macska legyen.
   - Az osztálynak legyen egy *eszik* metódusa, ami egy double értéket vár (étel mennyisége), és egy boolean-el tér vissza (sikeres volt -e az etetés). Ha a macska éhes, az etetés sikeres, és a súlya nőjön az étel mennyiségével. A macska ezután ne legyen éhes. Ha a macska nem éhes, az etetés nem sikeres.
   - Az osztálynak legyen egy void *futkos* metódusa, ami nem vár paramétert. A macska súlya csökkenjen 0.1-el, és ha nem volt éhes, akkor éhezzen meg.
   - Készíts toString metódust az osztályhoz.
   - A *main* metódusban hozz létre két macskát a két különböző konstruktorral, és próbáld meg megetetni őket. Az etetés sikerességéről írj információt konzolra.
   - Mindkét macska futkosson, és utána írd ki szövegesen az objektumokat.

```
1  public class Macska{
2    String nev;
3    double suly;
4    boolean ehes;
5
6    public Macska(String nev, double suly, boolean ehes){
7      this.nev = nev;
8      this.suly = suly;
9      this.ehes = ehes;
10   }
11
12   public Macska(String nev, double suly){
13     this(nev, suly, true);
14   }
15
```

```
16    public boolean eszik(double etel){
17      if(ehes){
18        this.suly += etel;
19        ehes = false;
20        return true;
21      }
22      return false;
23    }
24
25    public void futkos(){
26      if(suly-0.1>0)
27        suly -= 0.1;
28      ehes = false;
29    }
30
31    public String toString(){
32      return nev + " macska, " + suly + " kg, " + (ehes?"ehes":"nem
          ehes");
33    }
34
35
36    public static void main(String[] args){
37      Macska m1 = new Macska("Kormi",4.0,false);
38      Macska m2 = new Macska("Cirmi",3.5);
39
40      if(m1.eszik(0.2))
41        System.out.println(m1.nev + " macska evett");
42      else
43        System.out.println(m1.nev + " macska nem volt ehes");
44      if(m2.eszik(0.2))
45        System.out.println(m2.nev + " macska evett");
46      else
47        System.out.println(m1.nev + " macska nem volt ehes");
48
49      m1.futkos();
50      System.out.println(m1);
51      m2.futkos();
52      System.out.println(m2);
53
54    }
55  }
```

Macska megoldása

2. Írj osztályt, ami egy *Szamitogep* objektumot valósít meg.
   - A számítógép adattagjai a következőek legyenek: szabad memória MB-ban (double), be van -e kapcsolva (boolean).
   - Készíts két konstruktort is az osztályhoz. Az egyik általános legyen, ami minden adattagot a paraméterlistából állít be, a másik egy alapértelmezett konstruktor legyen, ami 1024 MB memóriával, kikapcsolva hozza létre a gépet.
   - Az osztálynak legyen egy void *kapcsol* metódusa, ami nem vár paramétert. Ha a gép ki van kapcsolva, akkor kapcsolja be, egyébként kapcsolja ki.
   - Az osztálynak legyen egy boolean *programMasol* metódusa, ami egy program méretét várja paraméternek MB-ban (double). Ha a program ráfér még a gépre, és a gép be van kapcsolva, úgy csökkenjen a szabad memória a program méretével. A metódus térjen vissza boolean változóval, hogy sikeres volt -e a másolás.
   - Készíts toString metódust az osztályhoz.
   - A *main* metódusban hozz létre két számítógépet a fenti konstruktorokkal. Mindkét gép kikapcsolt állapotban kezdjen. Az alapértelmezett gépet kapcsold be, és másold rá először 800 MB, aztán 400 MB programot. A másik gépre másolj 1 MB programot. A másolások eredményeit írd ki.
   - Mindkét objektumot írd ki szövegesen.

```
1  public class Szamitogep{
2    double memoria;
3    boolean bekapcsolva;
4
5    public Szamitogep(double memoria, boolean bekapcsolva){
6      this.memoria = memoria;
7      this.bekapcsolva = bekapcsolva;
8    }
9
10   public Szamitogep(){
11     this(1024, false);
12   }
13
14   public void kapcsol(){
15     bekapcsolva = !bekapcsolva;
16   }
17
18   public boolean programMasol(double meret){
19     if(bekapcsolva && meret<=memoria){
20       memoria -= meret;
```

```
21        return true;
22      }
23      return false;
24    }
25
26    public String toString(){
27      return (bekapcsolva?"bekapcsolt ":"kikapcsolt ") + "szamitogep
            " + memoria + " MB szabad memoriaval";
28    }
29
30
31    public static void main(String[] args){
32      Szamitogep sz1 = new Szamitogep();
33      Szamitogep sz2 = new Szamitogep(2048,false);
34
35      sz1.kapcsol();
36
37      if(sz1.programMasol(800)){
38        System.out.println("Program masolasa sikeres.");
39      }
40      else{
41        System.out.println("ERROR: Nem tudok programot masolni.");
42      }
43
44      if(sz1.programMasol(400)){
45        System.out.println("Program masolasa sikeres.");
46      }
47      else{
48        System.out.println("ERROR: Nem tudok programot masolni.");
49      }
50
51      if(sz2.programMasol(1)){
52        System.out.println("Program masolasa sikeres.");
53      }
54      else{
55        System.out.println("ERROR: Nem tudok programot masolni.");
56      }
57
58      System.out.println(sz1);
59      System.out.println(sz2);
60    }
```

```
61 }
```

Szamitogep megoldása

3. Írj osztályt, ami egy *Hallgato* objektumot valósít meg.
   - A hallgató adattagjai a következőek legyenek: azonosító (String), évfolyam (int), kreditszám (int).
   - Két konstruktort is készíts az osztályhoz. Az egyik általános legyen, ami minden adattagot a konstruktor paraméterlistájából állít be, illetve egy másik, ami az első adattagot a konstruktor paraméterlistájából kapja, évfolyama 1 és kreditszáma 0 legyen.
   - Az osztálynak legyen egy void *targyFelvesz* metódusa, amivel egy int paramétert (tárgy kreditértéke) kér. A hallgató kreditszáma nőjön a kapott értékkel.
   - Az osztálynak legyen egy boolean *vizsgazik* metódusa, ami nem vár paramétert. Ha a hallgatónak 0-nál több kreditje van, akkor a sikeres a vizsga: a következő évfolyamba lép, és nullázódik a kreditszáma. Egyébként a vizsga sikertelen.
   - Készíts toString metódust az osztályhoz.
   - A *main* metódusban hozz létre két hallgatót a két különböző konstruktorral. Az egyikük vegyen fel tárgyat, majd vizsgáztasd őket. A vizsga sikerességéről írj információt konzolra.
   - Ezután mindkét hallgatót írd ki szövegesen.

```
1
2  public class Hallgato{
3    String azonosito;
4    int evfolyam, kredit;
5
6    public Hallgato(String azonosito, int evfolyam, int kredit){
7      this.azonosito = azonosito;
8      this.evfolyam = evfolyam;
9      this.kredit = kredit;
10   }
11
12   public Hallgato(String azonosito){
13     this(azonosito,1,0);
14   }
15
16   public void targyFelvesz(int targyKredit){
17     kredit += targyKredit;
18   }
19
20   public boolean vizsgazik(){
21     if(kredit > 0){
```

```
22        evfolyam++;
23        kredit = 0;
24        return true;
25      }
26      return false;
27    }
28
29    public String toString(){
30      return "azonosito: " + azonosito + ", evfolyam: " + evfolyam +
            ", kredit: " + kredit;
31    }
32
33    public static void main(String[] args){
34      Hallgato h1 = new Hallgato("abc123");
35      Hallgato h2 = new Hallgato("szte.sze",5,0);
36
37      h1.targyFelvesz(5);
38      if(!h1.vizsgazik())
39        System.out.println(h1.azonosito + " hallgatonak sikertelen a
              vizsgaja.");
40      else
41        System.out.println(h1.azonosito + " hallgato sikeresen
             vizsgazott!");
42
43      if(!h2.vizsgazik())
44        System.out.println(h2.azonosito + " hallgatonak sikertelen a
              vizsgaja.");
45      else
46        System.out.println(h2.azonosito + " hallgato sikeresen
             vizsgazott!");
47
48      System.out.println(h1);
49      System.out.println(h2);
50    }
51  }
```

Hallgato megoldása

4. Írj osztályt, ami egy *Torta* objektumot valósít meg.
   - A torta adattagjai a következőek legyenek: emeletek száma (int), meg van -e kenve krémmel (boolean).
   - Készíts két konstruktort is az osztályhoz. Az egyik általános legyen, ami minden adat-

tagot paraméterlistából állít be, a másik egy alapértelmezett konstruktor legyen, ami 1 emeletes, krém nélküli tortát hoz létre.

- Az osztálynak legyen egy void *ujEmelet* metódusa, ami nem vár paramétert, és egy új emeletet rak a tortára.
- Az osztálynak legyen egy boolean *kremmelMegken* metódusa, ami nem vár paramétert. Ha a torta még nincs megkenve krémmel, úgy a metódus tegye ezt meg. Térjen vissza logikai értékkel attól függően, hogy sikerült -e.
- Készíts egy int típusú *mennyiKaloria* metódust az osztályhoz. A torta minden emelete 1000 kalória értékű, ha még krémmel is meg van kenve, akkor ennek a kétszerese.
- Készíts toString metódust az osztályhoz.
- A *main* metódusban hozz létre két tortát a két konstruktorral. Az alapértelmezett tortát kétszer is kend meg krémmel, ennek eredményét mindig írd konzolra. A másik tortára rakj egy emeletet.
- Mindkét objektumot írd ki szövegesen.

```
1  public class Torta{
2    int emelet;
3    boolean megkenve;
4
5    public Torta(int emelet, boolean megkenve){
6      this.emelet = emelet;
7      this.megkenve = megkenve;
8    }
9
10   public Torta(){
11     this(1,false);
12   }
13
14   public void ujEmelet(){
15     emelet++;
16   }
17
18   public boolean kremmelMegken(){
19     if(!megkenve){
20       megkenve = true;
21       return true;
22     }
23     return false;
24   }
25
26   public int mennyiKaloria(){
27     if(megkenve)
```

```
28        return 1000*emelet*2;
29      else
30        return 1000*emelet;
31    }
32
33    public String toString(){
34      return emelet + " emeletes torta" + (megkenve?" kremmel
           megkenve":"") + ", kaloriak: " + mennyiKaloria();
35    }
36
37
38    public static void main(String[] args){
39      Torta t1 = new Torta();
40      Torta t2 = new Torta(3,false);
41
42      for(int i=0;i<2;i++){
43        if(t1.kremmelMegken())
44          System.out.println("A " + t1 + " tortat megkentem.");
45        else
46          System.out.println("A " + t1 + " torta mar meg van kenve."
               );
47      }
48
49      t1.ujEmelet();
50
51      System.out.println(t1);
52      System.out.println(t2);
53    }
54  }
```

Torta megoldása

5. Írj osztályt, ami egy *Ember* objektumot valósít meg.
   - Az ember adattagjai a következőek legyenek: vezetéknév (String), keresztnév (String), születési év (int). Az adattagok csak ebből az osztályból legyen elérhetőek.
   - Készíts az osztályhoz konstruktor, ami paraméterek alapján állítja be az adattagokat.
   - Az osztálynak legyen egy *hogyHivjak* metódusa, ami visszaadja az ember telhes nevét (vezetéknév+keresztnév).
   - A *main* metódusban hozz létre egy ember objektumot, és írd ki a nevét.

```
1   public class Ember{
2     private String vNev, kNev;
```

```
  private int szulEv;

  public Ember(String vNev, String kNev, int szulEv){
    this.vNev = vNev;
    this.kNev = kNev;
    this.szulEv = szulEv;
  }

  public String hogyHivjak(){
    return vNev + " " + kNev;
  }

  public static void main(String[] args){
    Ember e = new Ember("Besenyo", "Pista", 1958);
    System.out.println(e.hogyHivjak());
  }
}
```

Ember megoldása

6. Írj osztályt, ami egy *Harcos* objektumot valósít meg.
   - Az ember adattagjai a következőek legyenek: név (String), életerő (int), harci erő (int). Az adattagok csak ebből az osztályból legyen elérhetőek.
   - Készíts az osztályhoz konstruktor, ami paraméterek alapján állítja be az adattagokat.
   - Az osztálynak legyen egy boolean *harcol* metódusa, ami egy másik harcost kap paraméternek. A metódus mindkét harcos életerejét csökkentse a másik harcos harci erejével. Ha valamelyik harcos elveszti a harcot (életereje 0 alá csökkenne), a metódus térjen vissza igazzal, egyébként hamissal.
   - Készítsd el a megfelelő metódusokat az adattagok lekérdezéséhez és módosításához.
   - Készíts toString metódust mindkét osztályhoz.
   - A *main* metódusban hozz létre két harcos objektumot. Harcoljanak, amíg valamelyikük el nem veszti a harcot.

```
public class Harcos {
  private String nev;
  private int eletero;
  private int harciero;

  public Harcos(String nev, int eletero, int harciero){
    this.nev = nev;
    this.eletero = eletero;
    this.harciero = harciero;
```

```
10    }
11
12    public int getEletero(){
13      return eletero;
14    }
15
16    public int getHarciero(){
17      return harciero;
18    }
19
20    public void sebzodik(int eletero){
21      this.eletero -= eletero;
22    }
23
24    public boolean harcol(Harcos h){
25      this.sebzodik(h.getHarciero());
26      h.sebzodik(this.getHarciero());
27      if(this.getEletero()<1 || h.getEletero()<1)
28        return true;
29      return false;
30    }
31
32    public String toString(){
33      return nev + " (HE: " + getHarciero() + ", EE: " + getEletero
          ()+")";
34    }
35
36
37    public static void main(String[] args){
38      Harcos h1 = new Harcos("Arcsibald",100,10);
39      Harcos h2 = new Harcos("Boborjan",80,15);
40
41      int kor = 1;
42      while(!h1.harcol(h2)){
43        System.out.println(kor + ". kor");
44        kor++;
45        System.out.println(h1);
46        System.out.println(h2);
47      }
48
49      if(h1.getEletero()<1 && h2.getEletero()<1)
```

```
50        System.out.println("Mindketten vesztettek.");
51      else if(h1.getEletero()<1)
52        System.out.println("Nyertes :" + h2);
53      else
54        System.out.println("Nyertes :" + h1);
55
56    }
57  }
```

Harcos megoldása

5 Objektumok metódusai

1. Írj osztályt, ami egy Stringet tárol. Ezt példányosításkor a konstruktorban kapott paraméter segítségével állítsa be. Az osztálynak legyen egy saját függvénye, ami egy karakter paramétert kér. A függvény adjon vissza egy olyan Stringet, amiben a tárolt adattag minden olyan karakterét, ami megegyezik a paraméterrel, '?' karakterre cseréli.

```
public class Feladat3_1{
  String word;
  public Feladat3_1(String word){
    this.word=word;
  }
  public String exchange(char c){
    char[] array=word.toCharArray();
    for(int i=0;i<array.length;i++){
      if(array[i]==c){
        array[i]='?';
      }
    }
    return new String(array);
  }
}
```

Feladat_3_1 megoldása.

2. Írj osztályt, ami egy Stringet tárol. Ezt példányosításkor a konstruktorban kapott paraméter segítségével állítsa be. Az osztálynak legyen egy saját függvénye, ami egy karakter paramétert kér. A függvény adjon vissza egy olyan Stringet, amiben a tárolt adattag minden olyan karakterét, ami nem egyezik meg a paraméterrel, '*' karakterre cseréli.

```
public class Feladat3_2{
  String word;
  public Feladat3_2(String word){
    this.word=word;
  }
```

```
6     public String exchange(char c){
7       char[] array=word.toCharArray();
8       for(int i=0;i<array.length;i++){
9         if(array[i]!=c){
10          array[i]='*';
11        }
12      }
13      return new String(array);
14    }
15  }
```

Feladat_3_2 megoldása.

3. Írj osztályt, ami egy Stringet tárol. Ezt példányosításkor a konstruktorban kapott paraméter segítségével állítsa be. Az osztálynak legyen egy saját függvénye. A függvény adjon vissza egy olyan Stringet, amiben a tárolt adattag minden betű karakterét '?' karakterre, és minden nembetű karakterét '*' karakterre cseréli.

```
1   public class Feladat3_3{
2     String word;
3     public Feladat3_3(String word){
4       this.word=word;
5     }
6     public String exchange(){
7       char[] array=word.toCharArray();
8       for(int i=0;i<array.length;i++){
9         if(Character.isLetter(array[i])){
10          array[i]='?';
11        }else{
12          array[i]='*';
13        }
14      }
15      return new String(array);
16    }
17  }
```

Feladat_3_3 megoldása.

4. Írj osztályt, ami tetszőleges számú Stringet (egy String tömböt) tárol. Ezt példányosításkor a konstruktorban kapott paraméter segítségével állítsa be. Az osztálynak egy saját függvénye legyen, ami azt adja vissza, hogy összesen hány karakterből állnak a tárolt Stringek.

```java
public class Feladat3_4{
  String[] words;
  public Feladat3_4(String[] words){
    this.words=words;
  }
  public int counter(){
    int count=0;
    for(int i=0;i<words.length;i++){
      count+=words[i].length();
    }
    return count;
  }
}
```

Feladat_3_4 megoldása.

5. Írj osztályt, ami tetszőleges számú Stringet (egy String tömböt) tárol. Ezt példányosításkor a konstruktorban kapott paraméter segítségével állítsa be. Az osztály egy saját függvénye adjon vissza egy olyan Stringet, ami az adattagban lévő Stringek összefűzése (abban a sorrendben, ahogy a tömbben szerepelnek). Két módon oldd meg a feladatot: a Stringeket segédtömb használatával, vagy anélkül fűzd össze.

```java
public class Feladat3_5{
  String[] words;
  public Feladat3_5(String[] words){
    this.words=words;
  }
  public String concatenatearray(){
    int count=0;
    for(int i=0;i<words.length;i++){
      count+=words[i].length();
    }
    char[] array=new char[count];
    int index=0;
    for(int i=0;i<words.length;i++){
      for(int j=0;j<words[i].length();j++){
        array[index]=words[i].charAt(j);
        index++;
      }
    }
    return new String(array);
  }
```

```
21    public String concatenatewithoutarray(){
22      String result="";
23      for(int i=0;i<words.length;i++){
24        result=result+words[i];
25      }
26      return result;
27    }
28  }
```

Feladat_3_5 megoldása.

6. Írj osztályt, ami két Stringet tárol. Ezeket példányosításkor a konstruktorban kapott paraméterek segítségével állítsa be. Az osztály egy saját függvénye adjon vissza egy olyan Stringet, amiben az első String után illeszti a második String fordítottját. Két módon oldd meg a feladatot: a Stringeket segédtömb használatával, vagy anélkül illeszd egymás után.

```
1   public class Feladat3_6{
2     String word1;
3     String word2;
4     public Feladat3_6(String word1, String word2){
5       this.word1=word1;
6       this.word2=word2;
7     }
8     public String copyarray(){
9       char[] result= new char[word1.length()+word2.length()];
10      int index=0;
11      for(int i=0;i<word1.length();i++){
12        result[index]=word1.charAt(i);
13        index++;
14      }
15      for(int i=word2.length()-1;i>=0;i--){
16        result[index]=word2.charAt(i);
17        index++;
18      }
19      return new String(result);
20    }
21    public String copywithoutarray(){
22      String result=new String(word1);
23      for(int i=word2.length()-1;i>=0;i--){
24        result=result+word2.charAt(i);
25      }
26      return result;
```

```
27    }
28  }
```

Feladat_3_6 megoldása.

7. Írj osztályt, ami két Stringet tárol. Ezeket példányosításkor a konstruktorban kapott paraméterek segítségével állítsa be. Az osztály egy saját függvénye adjon vissza egy olyan Stringet, ami összefésüli a két tárolt Stringet az alábbi sorrendben: E1,M1,E2,M2... (ahol pl. E1 az első szó első karaktere, M2 a második szó második karaktere). Ha valamelyik szó hosszabb, mint a másik, úgy annak plusz karakterei az új szó végére kerülnek az eredeti sorrendjükben. Két módon oldd meg a feladatot: a Stringeket segédtömb használatával, vagy anélkül fésüld össze.

```
1  public class Feladat3_7{
2    String word1;
3    String word2;
4    public Feladat3_7(String word1, String word2){
5      this.word1=word1;
6      this.word2=word2;
7    }
8    public String alternatearray(){
9      char[] result = new char[word1.length()+word2.length()];
10     int index=0;
11     int length;
12     if(word1.length()>=word2.length()){
13       length=word1.length();
14     }else{
15       length=word2.length();
16     }
17     for(int i=0;i<length;i++){
18       if(i<word1.length()){
19         result[index]=word1.charAt(i);
20         index++;
21       }
22       if(i<word2.length()){
23         result[index]=word2.charAt(i);
24         index++;
25       }
26     }
27     return new String(result);
28   }
29   public String alternatewithoutarray(){
30     String result="";
```

```
31      int length;
32      if(word1.length()>=word2.length()){
33        length=word1.length();
34      }else{
35        length=word2.length();
36      }
37      for(int i=0;i<length;i++){
38        if(i<word1.length()){
39          result+=word1.charAt(i);
40        }
41        if(i<word2.length()){
42          result+=word2.charAt(i);
43        }
44      }
45      return result;
46    }
47  }
```

Feladat_3_7 megoldása.

8. Írj osztályt, ami egy két dimenziós karakter tömböt tárol. Ezt a konstruktorban kapott paraéter alapján állítsd be. Az osztálynak egy saját függvénye legyen, ami egy karaktert kap paraméternek. A függvény keresse meg, hogy a tömb sor szerinti bejárásakor melyik (i,j) indexen fordul elő a kapott karakter, és ezzel az index-párral térj vissza.

```
1   public class Feladat3_8{
2     char[][] array;
3     public Feladat3_8(char[][] array){
4       this.array=array;
5     }
6     public int[] finder(char c){
7       int[] result=new int[2];
8       for(int i=0;i<array.length;i++){
9         for(int j=0;j<array[i].length;j++){
10          if(array[i][j]==c){
11            result[0]=i;
12            result[1]=j;
13            return result;
14          }
15        }
16      }
17      return result;
18    }
```

```
19 }
```

Feladat_3_8 megoldása.

9. Írj osztályt, ami egy két dimenziós karakter tömböt tárol. Ezt a konstruktorban kapott paraméter alapján állítsd be. Az osztálynak egy saját függvénye legyen, ami egy karaktert kap paraméternek. A függvény keresse meg, hogy a tömb oszlop szerinti bejárásakor melyik (i,j) indexen fordul elő a kapott karakter, és ezzel az index-párral térj vissza.

```
1  public class Feladat3_9{
2    char[][] array;
3    public Feladat3_9(char[][] array){
4      this.array=array;
5    }
6    public int[] finder(char c){
7      int[] result=new int[2];
8      for(int i=0;i<array[0].length;i++){
9        for(int j=0;j<array.length;j++){
10         if(array[j][i]==c){
11           result[0]=j;
12           result[1]=i;
13           return result;
14         }
15       }
16     }
17     return result;
18   }
19 }
```

Feladat_3_9 megoldása.

10. Írj osztályt, ami egy két dimenziós int tömböt tárol. Ezt a konstruktorban kapott paraméter alapján állítsd be. Az osztálynak egy saját függvénye legyen. A függvény nézze meg, hogy a tárolt tömb sorminimumai közül melyik a legnagyobb, és ezzel az értékkel térjen vissza.

```
1  public class Feladat3_10{
2    int[][] array;
3    public Feladat3_10(int[][] array){
4      this.array=array;
5    }
6    public int minmax(){
7      int max=Integer.MIN_VALUE;
8      for(int i=0;i<array.length;i++){
```

```
      int min=Integer.MAX_VALUE;
      for(int j=0;j<array[i].length;j++){
        if(array[i][j]<min){
          min=array[i][j];
        }
      }
      if(min>max){
        max=min;
      }
    }
    return max;
  }
}
```

Feladat_3_10 megoldása.

11. Írj osztályt, ami egy két dimenziós int tömböt tárol. Ezt a konstruktorban kapott paraméter alapján állítsd be. Az osztálynak egy saját függvénye legyen. A függvény nézze meg, hogy a tárolt tömb sorösszegei közül melyik a legkisebb, és ezzel az értékkel térjen vissza.

```
public class Feladat3_11{
  int[][] array;
  public Feladat3_11(int[][] array){
    this.array=array;
  }
  public int sum(){
    int min=Integer.MAX_VALUE;
    for(int i=0;i<array.length;i++){
      int sum=0;
      for(int j=0;j<array[i].length;j++){
        sum+=array[i][j];
      }
      if(sum<min){
        min=sum;
      }
    }
    return min;
  }
}
```

Feladat_3_11 megoldása.

12. Írj osztályt, ami egy két dimenziós int tömböt tárol. Ezt a konstruktorban kapott paraméter alapján állítsd be. Az osztálynak egy saját függvénye legyen. A függvény adja össze

a tömb keretében található számokat, (első sor, utolsó sor, első oszlop, utolsó oszlop), és ezzel az értékkel térjen vissza.

```
public class Feladat3_12{
  int[][] array;
  public Feladat3_12(int[][] array){
    this.array=array;
  }
  public int bordersum(){
    int sum=0;
    for(int i=0;i<array[0].length;i++){
      sum+=array[0][i];
    }
    for(int i=0;i<array[array.length-1].length;i++){
      sum+=array[array.length-1][i];
    }
    for(int i=1;i<array.length-1;i++){
      sum+=array[i][0];
    }
    for(int i=1;i<array.length-1;i++){
      sum+=array[i][array[i].length-1];
    }
    return sum;
  }
}
```

Feladat_3_12 megoldása.

13. Írj osztályt, ami egy két dimenziós int tömböt tárol. Ezt a konstruktorban kapott paraméter alapján állítsd be. Az osztálynak egy saját függvénye legyen, ami két egész számot (i és j indexeket) vár paraméternek. A függvény döntse el, hogy a kapott (i,j) indexpár környezetében (az indexpár, és 8 szomszédja) hány 1-es érték található, és ezzel térjen vissza. Kezeld, hogy a tömbről kilógó szomszédokat ne vizsgálja.

```
public class Feladat3_13{
  int[][] array;
  public Feladat3_13(int[][] array){
    this.array=array;
  }
  public int neighbourssum(int firstindex, int secondindex){
    int sum=0;
    for (int i=Math.max(0,firstindex-1); i <= Math.min(firstindex
       +1,array.length-1);++i){
```

```
9           for (int j = Math.max(0,secondindex-1); j<=Math.min(
               secondindex+1,array[0].length-1);++j) {
10              if (!(i==firstindex && j==secondindex)){
11                  if(array[i][j]==1){
12                    sum+=array[i][j];
13                  }
14              }
15          }
16      }
17      return sum;
18    }
19  }
```

Feladat_3_13 megoldása.

6 Csomagok és láthatóságok

1. Készítsd el az alábbi feladatot:
   - Készíts egy *Egyetem* osztályt, és rakd az *intezmeny* csomagba. Az egyetemnek két String adattagja legyen: a neve, valamint a város, ahol található.
   - Készíts egy *Vegzettseg* osztályt, és rakd az *intezmeny* csomagba. Két adattagja legyen: milyen szakon (String), és milyen egyetemen szerezték a végzettséget.
   - Készíts egy *Oktato* osztályt, és rakd az *oktatas* csomagba. Egy oktatónak legyen neve (String) és végzettsége.
   - Készíts egy *Kurzus* osztályt, és rakd az *oktatas* csomagba.. A kurzusnak legyen neve (String), és oktatója.
   - A fenti összes objektumnak legyen továbbá paraméteres konstruktora, ami beállítja az adattagok értékeit, getter és setter metódusok az adattagokhoz, és toString metódus, ami szöveges formára hozza őket.
   - Írj main osztályt, és rakd a *futtat* csomagba. A main metódusban példányosíts egyetemeket, majd ezek segítségével hozz létre végzettségeket, majd oktatókat. Hozz létre egy statikus Kurzus objektumokat tároló tömböt is (tetszőleges mérettel), és a tömb minden elemébe példányosíts egy Kurzus objektumot. Írj ki minden kurzust konzolra.
   - Készíts metódust, ami egy város nevét kapja paraméterül. A metódus járja be a fenti kurzustömböt, és számold meg, hogy hány kurzus oktatója szerezte a végzettségét az adott városban. A metódus ezzel az eredménnyel térjen vissza.

```
1  package intezmeny;
2
3  public class Egyetem {
4
5      private String nev;
6      private String varos;
7
8      public Egyetem(String nev, String varos) {
9          this.nev = nev;
10         this.varos = varos;
11     }
12
```

```
13      public String getNev() {
14          return nev;
15      }
16
17      public String getVaros() {
18          return varos;
19      }
20
21      public void setVaros(String varos) {
22          this.varos = varos;
23      }
24
25      public void setNev(String nev) {
26          this.nev = nev;
27      }
28
29    @Override
30      public String toString() {
31          return nev + "," + varos;
32      }
33  }
```

Egyetem.java

```
1   package intezmeny;
2
3   public class Vegzettseg {
4
5       private String szak;
6       private Egyetem egyetem;
7
8       public Vegzettseg(String szak, Egyetem egyetem) {
9           this.szak = szak;
10          this.egyetem = egyetem;
11      }
12
13      public String getSzak() {
14          return szak;
15      }
16
17      public Egyetem getEgyetem() {
18          return egyetem;
```

```
19      }
20
21      public void setSzak(String szak) {
22          this.szak = szak;
23      }
24
25      public void setEgyetem(Egyetem egyetem) {
26          this.egyetem = egyetem;
27      }
28
29    @Override
30      public String toString() {
31          return szak + " (" + egyetem + ")";
32      }
33
34  }
```

Vegzettseg.java

```
1   package oktatas;
2
3   import intezmeny.Vegzettseg;
4
5   public class Oktato {
6
7       private String nev;
8       private Vegzettseg vegzettseg;
9
10      public Oktato(String nev, Vegzettseg vegzettseg) {
11          this.nev = nev;
12          this.vegzettseg = vegzettseg;
13      }
14
15      public String getNev() {
16          return nev;
17      }
18
19      public Vegzettseg getVegzettseg() {
20          return vegzettseg;
21      }
22
23      public void setNev(String nev) {
```

```
24          this.nev = nev;
25      }
26
27      public void setVegzettseg(Vegzettseg vegzettseg) {
28          this.vegzettseg = vegzettseg;
29      }
30
31    @Override
32      public String toString() {
33          return "Nev: " + nev + ", vegzettseg: " + vegzettseg;
34      }
35  }
```

Oktato.java

```
1   package oktatas;
2
3   public class Kurzus{
4
5       private String nev;
6       private Oktato oktato;
7
8       public Kurzus(String idopont, Oktato oktato){
9           this.nev = idopont;
10          this.oktato = oktato;
11      }
12
13      public String getNev(){
14          return nev;
15      }
16
17      public Oktato getOktato(){
18          return oktato;
19      }
20
21      public void setNev(String nev){
22          this.nev = nev;
23      }
24
25      public void setOktato(Oktato oktato){
26          this.oktato = oktato;
27      }
```

```
28
29    @Override
30      public String toString(){
31          return "Kurzus neve: " + nev + ", oktato adatai: " +
                oktato;
32      }
33
34  }
```

Kurzus.java

```
1   package futtat;
2
3   import oktatas.*;
4   import intezmeny.*;
5
6   public class OktMain {
7
8       static Kurzus[] kurzusok = new Kurzus[4];
9
10      static int varosbanVegzettek(String varos) {
11          int db = 0;
12          for (int i = 0; i < kurzusok.length; i++) {
13              if(kurzusok[i].getOktato().getVegzettseg().getEgyetem
                    ().getVaros().equals(varos))
14                  db++;
15          }
16          return db;
17      }
18
19      public static void main(String[] args) {
20
21          if (args.length == 0) {
22              System.out.println("Egy varos nevet kerem bemenetnek."
                    );
23              System.exit(0);
24          }
25          Egyetem szte = new Egyetem("SZTE", "Szeged");
26          Egyetem elte = new Egyetem("ELTE", "Budapest");
27
28          Vegzettseg v1 = new Vegzettseg("progmat", elte);
29          Vegzettseg v2 = new Vegzettseg("progmat", szte);
```

```
30
31          Oktato o1 = new Oktato("A. Arcsibald", v1);
32          Oktato o2 = new Oktato("B. Boborjan", v2);
33
34          kurzusok[0] = new Kurzus("ProgAlap", o1);
35          kurzusok[1] = new Kurzus("OOP", o2);
36          kurzusok[2] = new Kurzus("Kalkulus", o1);
37          kurzusok[3] = new Kurzus("Unity", o2);
38
39          for (int i = 0; i < kurzusok.length; i++)
40              System.out.println(kurzusok[i]);
41
42          System.out.println("Ennyien vegeztek " + args[0] + "
               varosban: " + varosbanVegzettek(args[0]));
43      }
44
45  }
```

OktMain.java

2. Készítsd el az alábbi feladatot:
   - Készíts egy *Pont* osztályt, és rakd a *_2d* csomagba. A 2D pontnak két int koordinátája legyen. Tárolja továbbá statikus konstansként, hogy mik a koordináták alapértelmezett értékei.
   - Két konstruktort is készíts az osztályhoz. Az egyik paraméterekből állítsa be az adattagokat, a másik az alapértelmezett értékeket használja.
   - Minden adattaghoz vegyél fel getter és setter metódusokat, illetve legyen az osztálynak toString metódusa is.
   - Készíts egy *Kor* osztályt a *_2d* csomagba. A körnek két adattagja legyen: középpont (Pont) és sugár (double).
   - Három konstruktort készíts a körhöz:
     - Az első egy pontot és egy sugarat kérjen paraméternek, és ez alapján állítsa be az adattagokat.
     - A második egy pont koordinátáit (int alakban), valamint a sugarat kérje paraméternek, és ez alapján állítsa be az adattagokat.
     - A harmadik ne kérjen paramétert, és az alapértelmezett koordinátákba kerüljön a középpont, 1-es sugárral.
   - Az osztálynak legyen egy statikus int adattagja is, ami számolja, hogy összesen hány kör lett már létrehozva a program futása során. Ezt növeld minden egyes létrehozáskor.
   - Írj getter és setter metódusokat az adattagokhoz. A pont esetén overloading segítségével oldd meg, hogy pont objektumot, vagy int koordinátákat is elfogadjon a setter.
   - Készíts két metódust, ami visszaadja a kör kerületét és területét.

- Készíts statikus metódust, ami visszaadja a létrehozott köröket számláló változó értékét.
- Készíts egy *eltol* metódust, ami a kör középpontját megadott x és y irányba arrébb tolja.
- Készíts toString metódust, ami kiírja a kör fontosabb információit.
- Készíts egy *Pont* osztályt, és rakd a *_3d* csomagba. A 3D pontnak három int koordinátája legyen. Tárolja továbbá statikus konstansként, hogy mik a koordináták alapértelmezett értékei.
- A fentiekhez hasonlóan szintén két konstruktort készíts az osztályhoz. Az egyik paraméterekből állítsa be az adattagokat, a másik az alapértelmezett értékeket használja.
- Minden adattaghoz vegyél fel getter és setter metódusokat, illetve legyen az osztálynak toString metódusa is.
- Készíts egy *Gomb* osztályt a *_3d* csomagba. A gömbnek két adattagja legyen: középpont (Pont) és sugár (double).
- A körhöz hasonlóan készíts szintén három konstruktort a gömbnek:
  - Az első egy pontot és egy sugarat kérjen paraméternek, és ez alapján állítsa be az adattagokat.
  - A második egy pont koordinátáit (int alakban), valamint a sugarat kérje paraméternek, és ez alapján állítsa be az adattagokat.
  - A harmadik ne kérjen paramétert, és az alapértelmezett koordinátákba kerüljön a középpont, 1-es sugárral.
- Írj getter és setter metódusokat az adattagokhoz. A pont esetén overloading segítségével oldd meg, hogy pont objektumot, vagy int koordinátákat is elfogadjon a setter.
- Készíts két metódust, ami visszaadja a gömb felszínét és térfogatát.
- Készíts toString metódust, ami kiírja a kör fontosabb információit.
- Készíts egy futtatható osztályt a futtat csomagba. A main metódusban hozz létre két kört és egy gömböt, írasd ki, hogy hány kör létezik már, illetve told arrébb az egyik kör középpontját. Ezután írasd ki mindegyik objektumot.

```
1  package _2d;
2
3  public class Pont {
4
5      private int x;
6      private int y;
7
8      public static final int ORIGOX = 0;
9      public static final int ORIGOY = 0;
10
11     public Pont(int x, int y) {
12         this.x = x;
13         this.y = y;
14     }
```

```
15
16      public Pont() {
17          this(ORIGOX, ORIGOY);
18      }
19
20      public int getX() {
21          return x;
22      }
23
24      public int getY() {
25          return y;
26      }
27
28      protected void setX(int x) {
29          this.x = x;
30      }
31
32      protected void setY(int y) {
33          this.y = y;
34      }
35
36      @Override
37      public String toString() {
38          return "(" + x + "," + y + ")";
39      }
40
41  }
```

_2d.Pont.java

```
1   package _2d;
2
3   public class Kor {
4
5       private Pont p;
6       private double r;
7
8       private static int countKor = 0;
9
10      public Kor(Pont p, double r) {
11          this.p = p;
12
```

```
13          if (r > 0) {
14              this.r = r;
15          } else {
16              this.r = 0;
17          }
18          countKor++;
19      }
20
21      public Kor(int x, int y, double r) {
22          this(new Pont(x, y), r);
23      }
24
25      public Kor() {
26          this(new Pont(), 1);
27      }
28
29      public static int getKorokSzama() {
30          return countKor;
31      }
32
33      public double getR() {
34          return r;
35      }
36
37      protected void setR(double r) {
38          if (r > 0) {
39              this.r = r;
40          } else {
41              this.r = 0;
42          }
43      }
44
45      public Pont getP() {
46          return p;
47      }
48
49      public void setP(Pont p) {
50          this.p = p;
51      }
52
53      public void setP(int x, int y) {
```

```
54          p = new Pont(x, y);
55      }
56
57      public double getKerulet() {
58          return 2 * r * Math.PI;
59      }
60
61      public double getTerulet() {
62          return r * r * Math.PI;
63      }
64
65      public void eltol(int x, int y) {
66          p.setX(p.getX() + x);
67          p.setY(p.getY() + y);
68      }
69
70      public String toString() {
71          return "p: " + p + ", r: " + r + ", kerulet: " +
               getKerulet() + ", terulet: " + getTerulet();
72      }
73  }
```

Kor.java

```
1   package _3d;
2
3   public class Pont {
4
5       private int x, y, z;
6
7       public static final int ORIGOX = 0;
8       public static final int ORIGOY = 0;
9       public static final int ORIGOZ = 0;
10
11      public Pont(int x, int y, int z) {
12          this.x = x;
13          this.y = y;
14          this.z = z;
15      }
16
17      public Pont() {
18          this(ORIGOX, ORIGOY, ORIGOZ);
```

```
19      }
20
21      public int getX() {
22          return x;
23      }
24
25      public int getY() {
26          return y;
27      }
28
29      public int getZ() {
30          return z;
31      }
32
33      protected void setX(int x) {
34          this.x = x;
35      }
36
37      protected void setY(int y) {
38          this.y = y;
39      }
40
41      protected void setZ(int z) {
42          this.z = z;
43      }
44
45      public String toString() {
46          return "(" + x + "," + y + "," + z + ")";
47      }
48  }
```

_3d.Pont.java

```
1   package _3d;
2
3   public class Gomb {
4
5       private Pont p;
6       private double r;
7
8       public Gomb(Pont p, double r) {
9           this.r = r;
```

```
10          this.p = p;
11      }
12
13      public Gomb(int x, int y, int z, double r) {
14          this(new Pont(x, y, z), r);
15      }
16
17      public Gomb() {
18          this(new Pont(Pont.ORIGOX, Pont.ORIGOY, Pont.ORIGOZ), 1);
19      }
20
21      public Pont getP() {
22          return p;
23      }
24
25      public double getR() {
26          return r;
27      }
28
29      public double getFelszin() {
30          return 4 * r * r * Math.PI;
31      }
32
33      public double getTerfogat() {
34          return 4 * Math.pow(r, 3) * Math.PI / 3;
35      }
36
37      protected void setR(double r) {
38          this.r = r;
39      }
40
41      protected void setP(Pont p) {
42          this.p = p;
43      }
44
45      protected void setP(int x, int y, int z) {
46          p = new Pont(x, y, z);
47      }
48
49      @Override
50      public String toString() {
```

```
51          return "p: " + p + ", r: " + r + ", felszin: " +
                getFelszin() + ", terfogat: " + getTerfogat();
52      }
53  }
```

Gomb.java

```
1   package futtat;
2
3   import _2d.Kor;
4   import _3d.Gomb;
5
6   public class SikMain {
7
8
9       public static void main(String[] args) {
10          Kor k1 = new Kor(1,1,3.0);
11          System.out.println(k1);
12          Kor k2 = new Kor();
13          System.out.println(k2);
14          System.out.println("Ennyi korom van: " + Kor.getKorokSzama
                ());
15
16          Gomb g = new Gomb(1,2,3, 3.0);
17          System.out.println(g);
18
19          k2.eltol(3, 0);
20          System.out.println(k2);
21      }
22
23  }
```

SikMain.java

3. Oldd meg az alábbi feladatot:
   - Írj Pont osztályt, ami egy 2D-s pontot valósít meg, és helyezd el a geom csomagba.
     - Egy pontról el kell tárolni x és y koordinátáját, valamint típusát (origó, vagy egyéb pont).
     - Hozz létre két konstans adattagot is, mely az origó alapértelmezett x és y koordinátájának értékét tartalmazza (ezek értéke legyen 0 mindkét esetben 0)
     - Írj az osztálynak konstruktort, ami az x illetve y koordinátát kapja meg, és ezek alapján állítja be a mezők értékét (és eldönti azt is, hogy a pont origó -e).

  - Az osztályt lehessen példányosítani paraméterek megadása nélkül is, ilyenkor a koordináták az oriógval egyezzenek meg, és a mezők is ez alapján legyenek beállítva.
- Írj egy Teglalap osztályt is a geom csomagba téglalapok megvalósítására.
  - Egy téglalapnak van magassága, szélessége, valamint egy 4 elemű tömbben tárolja a 4 csúcsát.
  - A konstruktora kérje be a magasságot és a szélességet, valamint a bal felső csúcspontot. Ezeke alapján állítsa be a mezők értékét, és a csúcsok tömbjét is töltse ke (bal fent, jobb fent, bal lent, jobb lent sorrendben). A magasságot és a szélességet ne tárold le!
  - Hozz létre egy getKerulet metódust, ami kiszámolja a téglalap kerületét.
  - Hozz létre egy getTerulet metódust, ami kiszámolja a téglalap területét.
  - Az osztály legyen String formára alakítható a következő módon: "Teglalap - kerulet: K, terulet: T"
- Írj egy, a geom csomagon kívül futtatható osztályt, ebben:
  - Hozz létre egy Teglalap objektumot.
  - Hozz létre egy Pont objektumot, és ennek segítségével példányosíts egy téglalapot.
  - Írasd ki a Teglalap objektumot

```
1  package geom;
2
3  public class Pont {
4    private double x;
5    private double y;
6    private boolean origoE;
7    private final double ORIGOX=0.0;
8    private final double ORIGOY=0.0;
9
10   public Pont(){
11     this.x=0.0;
12     this.y=0.0;
13     this.origoE=true;
14   }
15
16   public Pont(double x, double y){
17     this.x=x;
18     this.y=y;
19     this.origoE= (y==ORIGOY && x==ORIGOX)? true: false;
20   }
21
22   public double getX() {
23     return x;
```

```
24    }
25
26    public void setX(double x) {
27      this.x = x;
28    }
29
30    public double getY() {
31      return y;
32    }
33
34    public void setY(double y) {
35      this.y = y;
36    }
37
38    public boolean isOrigoE() {
39      return origoE;
40    }
41
42    public void setOrigoE(boolean origoE) {
43      this.origoE = origoE;
44    }
45
46    public double getORIGOX() {
47      return ORIGOX;
48    }
49
50    public double getORIGOY() {
51      return ORIGOY;
52    }
53  }
```

Pont.java

```
1  package geom;
2
3  public class Teglalap {
4    double magassag;
5    double szelesseg;
6    Pont[] csucsok=new Pont[4];
7
8    public Teglalap(double magassag, double szelesseg, Pont balfelso
       ){
```

```
9       this.magassag=magassag;
10      this.szelesseg=szelesseg;
11      csucsok[0]=balfelso;
12      csucsok[1]=new Pont(balfelso.getX()+this.szelesseg,balfelso.
           getY());
13      csucsok[2]=new Pont(balfelso.getX(),balfelso.getY()-this.
           magassag);
14      csucsok[3]=new Pont(balfelso.getX()+this.szelesseg,balfelso.
           getY()-this.magassag);
15    }
16
17    public double getKerulet(){
18      return (this.magassag+this.szelesseg)*2;
19    }
20
21    public double getTerulet(){
22      return this.magassag*this.szelesseg;
23    }
24
25    @Override
26    public String toString() {
27      return "Teglalap kerulet=" + getKerulet() + ", terulet="+
           getTerulet();
28    }
29
30
31  }
```

Teglalap.java

```
1   package futtat;
2
3   import geom.Pont;
4   import geom.Teglalap;
5
6   public class Runnable {
7
8     public static void main(String[] args) {
9       Teglalap t=new Teglalap(10,5,new Pont(0,0));
10      System.out.println(t.toString());
11    }
12  }
```

Runnable.java

4. Oldd meg az alábbi feladatot:
   - Írj *Autobusz* osztályt, és helyezd a *vallalat* csomagba.
     - Az autóbuszról el kell tárolni, hogy hány szabad férőhelye van, mennyi a hatótávolsága (km-ben), és hány Ft-ba kerül kibérelni.
     - Az autóbuszhoz tartozzon egy maximális férőhely konstans is.
     - Írj 3 paraméteres konstruktort, ami a fenti adatokat kapja meg, és azok alapján állítja be az adattagokat! (a busz kezdetben üres, tehát a szabad és a maximális férőhelyek száma is ugyanaz az érték lesz)
     - Írj a busznak default konstruktort is, ami minden értéket -1re állít.
     - Írj egy *utasHozzaad* metódust, amely paraméterben kapja, hogy hány utast szeretnénk az autóbuszba beültetni, és igaz vagy hamis értékkel tér vissza annak alapján, hogy volt-e elég férőhely a buszban! Ha elég volt a férőhely, úgy csökkentsd a szabad férőhelyek számát is! Egyes csoportok csak együtt hajlandóak utazni, vagyis ha nem sikerül hozzáadni az egész csoportot, akkor senki nem kerül fel a buszra.
     - Írj minden adattaghoz *getter*, valamint *setter* függvényeket, ahova lehet!
     - Az osztály legyen szöveges formára alakítható, ami kiírja a az adattagok értékét (pl. "Busz X ferohellyel es Y km hatotavolsaggal)".
   - Írj *UtazasiIroda* osztályt, és helyezd a *vallalat* csomagba.
     - Minden irodának el kell tárolni, hogy hány db autóbusszal rendelkezik, valamint egy tömbben tárolja *Autobusz* objektumait is.
     - Írj konstruktort, ami egy az iroda maximális buszszámát kapja paraméterül, a rendelkezésre álló buszok számát 0-ra állítja, a buszokat tároló tömböt pedig a paraméternek megfelelően hozza létre!
     - Írj egy *buszHozzaad* metódust, amely a paraméterként kapott buszt hozzáadja az őket tároló tömbhöz, és megfelelően módosítja a darabszámot is! Ha nem adható már hozzá a tömbhöz az aktuális busz, úgy ne történjen semmi. A metódust térjen vissza logikai értékkel attól függően, hogy sikerült -e a hozzáadás.
     - Írj egy *buszBerel* metódust is, ami a megtenni kívánt km-t, valamint egy emberszámot kap paraméterül. A metódus adja vissza a legkisebb költségű buszt, amivel ezt az utat teljesíteni lehet. Egy utat csak olyan busz tud teljesíteni, aminek a hatótávolsága nem kisebb a megtenni kívánt km-nél, és van benne elég férőhely. A metódus a legolcsóbb ilyen busz objektummal térjen vissza. Ha nincs megfelelő busza a társaságnak, úgy egy alapértelmezett busz objektum legyen a visszatérési érték.
     - Oldd meg, hogy a *buszBerel* metódus csak km paraméter esetén is működjön. A metódus hasonlóan működjön, mint a fenti.
   - Írj egy futtatható osztályt a *futtat* csomagba, amiben:
     - Olvass be három számot parancssori argumentumból: az iroda buszainak a számát, egy utazó csoport létszámát, és a távolságot, ahova utazni akarnak.

– Hozz létre egy utazási irodát a kapott busszám paraméterrel, és töltsd fel véletlenszerűen létrehozott buszokkal. Egy busz férőhelyeinek száma 30-40 közötti, hatótávolsága 300-400 közötti legyen, költsége pedig a hatótávolság 1-2szerese.
– Minden létrehozott buszt írass ki konzolra.
– Döntsd el, hogy melyik a legolcsóbb busz, amivel a paraméterként kapott csoport utazni tud. Ha van ilyen, akkor írasd ki, és szállítsd fel a csoportot a buszra. Egyéb esetben írd ki, hogy "Nincs ilyen".

.

```
1  package vallalat;
2
3  public class Autobusz {
4    private int szabadferohely;
5    private int hatotav;
6    private int ar;
7    private final int maxferohely;
8
9    public Autobusz(){
10     this.szabadferohely=-1;
11     this.hatotav=-1;
12     this.ar=-1;
13     this.maxferohely=-1;
14   }
15
16   public Autobusz(int hatotav,int ar, int maxferohely){
17     this.szabadferohely=maxferohely;
18     this.hatotav=hatotav;
19     this.ar=ar;
20     this.maxferohely=maxferohely;
21   }
22
23   public boolean utasHozzaad(int utasokszama){
24     if(utasokszama>this.szabadferohely){
25       return false;
26     }else{
27       this.szabadferohely-=utasokszama;
28       return true;
29     }
30   }
31
32   public int getSzabadferohely() {
33     return szabadferohely;
```

```
34    }
35
36    public void setSzabadferohely(int szabadferohely) {
37      this.szabadferohely = szabadferohely;
38    }
39
40    public int getHatotav() {
41      return hatotav;
42    }
43
44    public void setHatotav(int hatotav) {
45      this.hatotav = hatotav;
46    }
47
48    public int getAr() {
49      return ar;
50    }
51
52    public void setAr(int ar) {
53      this.ar = ar;
54    }
55
56    public int getMaxferohely() {
57      return maxferohely;
58    }
59
60    @Override
61    public String toString() {
62      return "Busz "+this.maxferohely+" maximalis ferohellyel, "+
          this.hatotav+" hatotavval es "+this.szabadferohely+" szabad
           ferohellyel. Kiberelni "+ar+" Ft-ba kerul.";
63    }
64  }
```

Autobusz.java

```
1   package vallalat;
2
3   public class UtazasiIroda {
4     private int autobuszokszama;
5     private Autobusz[] buszok;
6
```

```
7   public UtazasiIroda(int maxbuszszam){
8     this.autobuszokszama=0;
9     this.buszok=new Autobusz[maxbuszszam];
10  }
11
12  public boolean buszHozzaad(Autobusz busz) {
13    if(autobuszokszama>buszok.length-1) {
14      return false;
15    }else{
16      buszok[autobuszokszama]=busz;
17      autobuszokszama++;
18      return true;
19    }
20  }
21
22  public Autobusz buszBerel(int km, int emberszam){
23    Autobusz eredmeny=new Autobusz();
24    int minar=Integer.MAX_VALUE;
25    for(int i=0;i<this.buszok.length;i++){
26      if(buszok[i].getAr()<minar && buszok[i].getHatotav()>=km &&
           buszok[i].getMaxferohely()>=emberszam){
27        eredmeny=buszok[i];
28        minar=buszok[i].getAr();
29      }
30    }
31    return eredmeny;
32  }
33
34  public Autobusz buszBerel(int km){
35    Autobusz eredmeny=new Autobusz();
36    int minar=Integer.MAX_VALUE;
37    for(int i=0;i<this.buszok.length;i++){
38      if(buszok[i].getAr()<minar && buszok[i].getHatotav()>=km){
39        eredmeny=buszok[i];
40        minar=buszok[i].getAr();
41      }
42    }
43    return eredmeny;
44  }
45 }
```

UtazasiIroda.java

```
package futtat;

import vallalat.Autobusz;
import vallalat.UtazasiIroda;

public class Runnable {
  public static void main(String[] args){
    if(args.length != 3){
      System.out.println("Nem megfelelo szamu parameter");
      System.exit(0);
    }
    int buszszam=Integer.parseInt(args[0]);
    int csoportletszam=Integer.parseInt(args[1]);
    int tavolsag=Integer.parseInt(args[2]);

    UtazasiIroda javatravel=new UtazasiIroda(buszszam);
    for(int i=0;i<buszszam;i++){
      int ferohely=(int)Math.round((Math.random()*10+30));
      int hatotav=(int)Math.round((Math.random()*100+300));
      int koltseg=hatotav*(int)Math.round(Math.random()*1+1);
      Autobusz busz=new Autobusz(hatotav,koltseg,ferohely);
      System.out.println(busz.toString());
      javatravel.buszHozzaad(busz);
    }
    Autobusz legolcsobb=javatravel.buszBerel(tavolsag,
       csoportletszam);
    if(legolcsobb.getAr()==-1 && legolcsobb.getHatotav()==-1 &&
       legolcsobb.getMaxferohely()==-1 && legolcsobb.
       getSzabadferohely()==-1){
      System.out.println("Nincs ilyen");
    }else {
      legolcsobb.utasHozzaad(csoportletszam);
      System.out.println("A legolcsobb busz " + legolcsobb.
         toString());
    }
  }
}
```

Runnable.java

5. Oldd meg az alábbi feladatot:
   - Írj Butorlap osztályt, ami egy bútorlapot valósít meg, és helyezd el a butorgyar csomagba.
     - Egy bútorlapról el kell tárolni a hosszúságát, szélességét, valamint típust (ez tartólap és hátlap lehet).
     - létre két konstans adattagot is, mely a tartólap és hátlap négyzetméterenkénti árát tartalmazza (ez legyen 5000 ill. 500)
     - Írj az osztálynak konstruktort, ami a bútorlap típusát, valamint hosszúságát és szélességét kapja meg cm-ben, és ezek alapján állítja be a mezők értékét. (!!m-cm konverzió kell a négyzetméterenkénti ár miatt!!)
     - Írj egy arSzamol metódust is, ami a bútorlap árát számolja ki a méret és típus alapján.
   - Írj egy Butor osztályt is a butorgyar csomagba bútorok megvalósítására.
     - Egy bútor megadja, hogy mennyi lapot tartalmaz, valamint tárolja a bútorlapjait egy külön tömbben.
     - Legyen olyan konstruktora, ami kellően nagy méretű tömböt (pl. 100 elem) hoz létre a bútorlapoknak, valamint a bútorlapok számát állítsa 0-ra.
     - Hozz létre olyan lapHozzaad metódust, ami hozzáad egy bútorlapot a tömbhöz.
     - Hozz létre olyan arSzamol metódust, ami kiszámolja a tömbben lévő bútorlapok alapján a bútor teljes árát.
     - Az osztály legyen String formára alakítható a következő módon: "Butor - lapok: x, ar y Ft"
   - Írj egy, a butorgyar csomagon kívül futtatható osztályt, ebben:
     - Hozz létre egy Butor objektumot.
     - Hozz létre Butorlap objektumokat, és add hozzá a Butorhoz.
     - Írasd ki a Butor objektumot.

```
1  package butorgyar;
2
3  public class Butorlap {
4    private double hosszusag;
5    private double szelesseg;
6    private boolean tartolape;
7    private final int TARTOLAPAR=5000;
8    private final int HATLAPAR=500;
9
10   public Butorlap(boolean tartolape, double hosszusagcm, double
       szelessegcm){
```

```
11      this.tartolape=tartolape;
12      this.hosszusag=hosszusagcm/100;
13      this.szelesseg=szelessegcm/100;
14    }
15
16    public double arSzamol(){
17      if(tartolape){
18        return hosszusag*szelesseg*TARTOLAPAR;
19      }else {
20        return hosszusag*szelesseg*HATLAPAR;
21      }
22    }
23
24    public double getHosszusag() {
25      return hosszusag;
26    }
27
28    public void setHosszusag(double hosszusag) {
29      this.hosszusag = hosszusag;
30    }
31
32    public double getSzelesseg() {
33      return szelesseg;
34    }
35
36    public void setSzelesseg(double szelesseg) {
37      this.szelesseg = szelesseg;
38    }
39
40    public boolean isTartolape() {
41      return tartolape;
42    }
43
44    public void setTartolape(boolean tartolape) {
45      this.tartolape = tartolape;
46    }
47
48    public int getTARTOLAPAR() {
49      return TARTOLAPAR;
50    }
51
```

```
52    public int getHATLAPAR() {
53      return HATLAPAR;
54    }
55  }
```

Butorlap.java

```
1   package butorgyar;
2
3   public class Butor {
4     int lapokszama;
5     Butorlap[] lapok;
6
7     public Butor(){
8       this.lapokszama=0;
9       this.lapok=new Butorlap[100];
10    }
11
12    public void lapHozzaad(Butorlap lap){
13      if(lapokszama<100){
14        this.lapok[lapokszama]=lap;
15        lapokszama++;
16      }
17    }
18
19    public double arSzamol(){
20      double ar=0;
21      for(int i=0;i<lapokszama;i++){
22        ar+=this.lapok[i].arSzamol();
23      }
24      return ar;
25    }
26
27    @Override
28    public String toString() {
29      return "Butor - lapok:"+lapokszama+", ar:"+arSzamol()+" Ft";
30    }
31  }
```

Butor.java

```
1   package futtat;
```

```
2
3  import butorgyar.Butor;
4  import butorgyar.Butorlap;
5
6  public class Runnable {
7
8    public static void main(String[] args) {
9      Butor szek=new Butor();
10     Butorlap a=new Butorlap(true, 35, 180);
11     Butorlap b=new Butorlap(false, 50, 190);
12     Butorlap c=new Butorlap(false, 76, 41);
13     szek.lapHozzaad(a);
14     szek.lapHozzaad(b);
15     szek.lapHozzaad(c);
16
17     System.out.println(szek.toString());
18   }
19 }
```

Runnable.java

6. Oldd meg az alábbi feladatot:
   - Készíts egy *Kartya* osztályt, ami játékkártyát valósít meg, és helyezd el a *zsuga* csomagba!
     - Egy kártyának van számértéke (2-14 között), és színe (piros, vagy fekete). Az adattagok csak ebből az osztályból legyenek elérhetőek!
     - Készíts 2 paraméteres konstruktort, ami beállítja a fenti értékeket. Ha nem megfelelő a számértéknek megadott paraméter, úgy a kártya értéke legyen 2!
     - Készíts default konstruktort, ami egy 0 értékű piros kártyalapot hoz létre!
     - Írj minden adattaghoz *getter* és *setter* függvényeket! Ha a számértéket nem megfelelő értékre akarják állítani, úgy azt hagyd figyelmen kívül.
     - Egy kártyalap legyen szöveges formára alakítható az alábbi módon: "szin ertek" (pl. piros 8).
   - Írj *Pakli* osztályt, és helyezd a *zsuga* csomagba!
     - Egy kártyapakli tömbben tárolja a benne lévő *Kartya* objektumokat. Legyen továbbá egy konstansa, ami a pakli joker színét tárolja. Az adattagok csak ebből az osztályból legyenek elérhetőek!
     - Kártyapaklit lehessen létrehozni maximális lapszámának és joker színének megadásával, valamint default konstruktorral is. A konstruktor inicializálja a lapokat tároló tömböt a megfelelő lapszámmal, és állítsa a joker konstanst a megadott értékre. Alapesetben 52 lapos egy pakli, és a fekete a joker. A konstruktor töltse fel a paklit véletlenszerűen generált *Kartya* objektumokkal (az érték és a szín is legyen véletlenszerű).

  - Készíts getter metódust, ami egy indexet vár, és a kártyákat tároló tömbből visszaadja az ott található kártyát! Ha az adott index nem megfelelő, úgy egy default kártyalappal térjen vissza.
  - Készíts getter metódust, ami a maximális lapszámot adja vissza.
  - Készíts a paklinak *osszesLap* metódust, ami végigmegy a benne tárolt kártyákon, és egyesével kiírja őket konzolra.
  - A pakli legyen szöveges formára alakítható az alábbi módon: "X lapos kartyapakli fekete/piros joker szinnel"
  - Készíts egy *blackJackLight* metódust, ami egy int paramétert vár. A paraméter megadja, hogy hány kártyát szeretnénk megnézni a pakli tetejéről, és a metódus visszaadja a kártyák értékének összegét. Egy kártyalap az értékének felét (felfele kerekítve) érik, ha a színe nem egyezik meg a pakli joker színével! Ha a paraméter nagyobb, mint a lapok aktuális száma, akkor csak annyi kártyát nézzünk meg, amennyit a tömb tárol.
  - Overloading segítségével oldd meg, hogy a *blackJackLight* metódus *Kartya* típusú paraméterrel is hívható legyen. Ebben az esetben addig nézz meg kártyákat a pakli tetejéről, amíg el nem éred a paraméterként kapott lapot. Ha nincs ilyen lap a pakliban, akkor nézd végig mindegyiket. A megnézett kártyák értékének összegét az előző ponthoz hasonlóan állapítsd meg, és ezzel térj vissza.
  - Készíts egy *egyszinuPakli* metódust, ami végigmegy a pakliban található kártyákon, és mindegyik színét a joker színre állítja.
- Készíts egy futtatható osztályt a *futtat* csomagba. A main metódusban:
  - Olvass be két egész számot parancssori argumentumból. Az első szám a pakli méretét, a második szám pedig a megnézendő lapok számát adja meg. Ellenőrizd, hogy kaptál -e két paramétert.
  - Hozz létre egy *Pakli* objektumot az első paraméter segítségével. Ha a paraméter nem pozitív szám, úgy a default konstruktort használd!
  - Írasd ki a pakli összes kártyáját a megfelelő metódus segítségével.
  - Játssz egy *blackJackLight* játékot a paklival (hívd meg a megfelelő metódust) a második paraméterrel, és írasd ki az eredményt.

```
1  package zsuga;
2
3  public class Kartya {
4
5      private int ertek;
6      private boolean isPiros;
7
8      public Kartya(int ertek, boolean isPiros) {
9          if (ertek >= 2 && ertek <= 14) {
10             this.ertek = ertek;
11         } else {
```

```
12              this.ertek = 2;
13          }
14          this.isPiros = isPiros;
15      }
16
17      public Kartya() {
18          this(0, true);
19      }
20
21      public int getErtek() {
22          return ertek;
23      }
24
25      public void setErtek(int ertek) {
26          if (ertek >= 2 && ertek <= 14) {
27              this.ertek = ertek;
28          }
29      }
30
31      public boolean isIsPiros() {
32          return isPiros;
33      }
34
35      public void setIsPiros(boolean isPiros) {
36          this.isPiros = isPiros;
37      }
38
39      @Override
40      public String toString() {
41          return (isPiros ? "piros " : "fekete ") + ertek;
42      }
43  }
```

Kartya.java

```
1   package zsuga;
2
3   public class Pakli {
4
5       private Kartya[] kartyak;
6       private final boolean JOKERPIROS;
7
```

```
8       public Pakli(int maxLapszam, boolean isPiros) {
9           JOKERPIROS = isPiros;
10          kartyak = new Kartya[maxLapszam];
11          genLapok();
12      }
13
14      public Pakli() {
15          this(52, false);
16      }
17
18      private void genLapok() {
19          for (int i = 0; i < kartyak.length; i++) {
20              int ertek = (int) Math.floor(Math.random() * 13 + 2);
21              boolean isPiros = Math.random() < 0.5;
22              kartyak[i] = new Kartya(ertek, isPiros);
23          }
24      }
25
26      public Kartya getKartya(int idx) {
27          if (idx > kartyak.length - 1 || idx < 0) {
28              return new Kartya();
29          } else {
30              return kartyak[idx];
31          }
32      }
33
34      public int getLapszam() {
35          return kartyak.length;
36      }
37
38      public String toString() {
39          return kartyak.length
40                  + " lapos pakli, joker: "
41                  + (JOKERPIROS ? "piros" : "fekete");
42      }
43
44      public int blackJackLight(int hanyLap) {
45          if (hanyLap > kartyak.length) {
46              hanyLap = kartyak.length;
47          }
48          if (hanyLap < 0) {
```

```
49              hanyLap = 0;
50          }
51
52          int ossz = 0;
53          for (int i = 0; i < hanyLap; i++) {
54              if (kartyak[i].isIsPiros() == JOKERPIROS) {
55                  ossz += kartyak[i].getErtek();
56              } else {
57                  ossz += Math.ceil(kartyak[i].getErtek() / 2.0);
58              }
59          }
60          return ossz;
61      }
62
63      public int blackJackLight(Kartya k) {
64          int idx = kartyak.length;
65
66          for (int i = 0; i < kartyak.length; i++) {
67              if (kartyak[i].isIsPiros() == k.isIsPiros()
68                      && kartyak[i].getErtek() == k.getErtek()) {
69                  idx = i + 1;
70                  break;
71              }
72          }
73
74          return blackJackLight(idx);
75      }
76
77      public void egyszinuPakli() {
78          for (int i = 0; i < kartyak.length; i++) {
79              kartyak[i].setIsPiros(JOKERPIROS);
80          }
81      }
82
83      public void osszesLap() {
84          for (int i = 0; i < kartyak.length; i++) {
85              System.out.println(kartyak[i]);
86          }
87      }
88  }
```

Pakli.java

```java
package futtat;

import zsuga.Pakli;

public class KartyasMain {

    public static void main(String[] args) {
        int lapszam = Integer.parseInt(args[0]);
        int hanyKartya = Integer.parseInt(args[1]);

        Pakli pakli;
        if (lapszam > 0) {
            pakli = new Pakli(lapszam, Math.random() < 0.5);
        } else {
            pakli = new Pakli();
        }

        pakli.osszesLap();
        System.out.println(pakli);

        System.out.println("A jatek eredmenye: "
                + pakli.blackJackLight(hanyKartya));
    }

}
```

KartyasMain.java

7 Öröklődés

1. Oldd meg az alábbi feladatot:
   - Írj egy *Arlap* interfészt, ami egy mennyibeKerul() metódust tartalmaz, mely egy lebegőpontos értékkel (az áru árával) tér vissza. Legyen továbbá egy CSESZEKAVE konstansa, ami egy kávé árát adja meg, ennek értéke 180.
   - Írj egy *Peksutemeny* absztrakt osztályt, ami implementálja az *Arlap* interfészt.
     - Az osztály a következő lebegőpontos adattagokkl rendelkezik: mennyiseg, alapar. Az alapár csak ebből az osztálból legyen látható, míg a mennyiség legyen látható a leszármazott osztályokban is (használd a lehető legszűkebb láthatóságot).
     - Az osztály rendelkezzen paraméteres konstruktorral, ami beállítja az adattagok értékeit. Legyen ezen felül egy *megkostol* public láthatóságú absztrakt függvénye, ami nem tér vissza értékkel. Az osztály valósítsa meg az implementált interfész metódusát. Egy péksütemény értéke az alapárának és a mennyiségének szorzatából számolható ki.
     - Az osztály legyen továbbá szöveges formára alakítható, kiírva az adattagok értékét. (pl. "X db - Y Ft", ahol Y azt jelenti, mennyibe kerül összesen).
   - Írj egy *Pogacsa* osztályt, ami a *Peksutemeny* leszármazottja.
     - Az osztálynak egy két paraméteres konstruktora legyen, ami a mennyiséget és az alapárat kéri el, majd állítja be.
     - Pogácsa megkóstolásakor mindig csökkenjen felére a mennyisége.
     - Az osztály legyen továbbá szöveges formára alakítható. Az objektum tulajdonságain kívül írja ki azt is, hogy Pogacsa osztályról van szó. (pl. "Pogacsa X db - Y Ft", ahol Y azt jelenti, mennyibe kerül összesen). Itt használd fel az ősosztály toString metódusát is!
   - Írj egy *Kave* osztályt, ami implementálja az *Arlap* interfészt.
     - Az osztály egy habosE private láthatóságú logikai adattaggal rendelkezik.
     - Az osztály rendelkezzen egy 1 paraméteres konstruktorral, ami beállítja az adattag értékét. A metódusai az alábbiak szerint legyenek megvalósítva: Egy rendes kávé ára annyi, amennyi az *Arlap* interfészben adott, habos kávé esetén ennek 1,5-szerese.
     - Az osztály legyen szöveges formára alakítható. (pl. "Habos/Nem habos kave - X Ft").
   - Írj egy *Pekseg* nevű, futtatható osztályt.
     - Az osztálynak legyen egy tárolója (tetszőleges kollekció, pl. lista), ahova Arlap típusú objektumokat tárol.

– Az osztály rendelkezzen egy *vasarlok* statikus metódussal, ami egy fájl elérési útját várja paraméterül, visszatérése pedig void. A metódus feladata, hogy a fájlból beolvasott sorokat feldolgozza, és létrehozzon belőlük *Pogacsa*, vagy *Kave* objektumpéldányokat. A fájl egy sorában az adott objektum tulajdonságai szerepelnek. A létrehozott objektumpéldányokat közös tárolóban tárold le.
– Készíts továbbá egy *etelLeltar* statikus metódust, ami végigmegy a tárolóban tárolt elemeken, és az összes Pogacsa típus objektum információit kiírja egy "leltar.txt" fájlba.
– Hívd meg a main függvényben sorban a fenti két metódust. A *vasarlok* metódus paraméterét parancssori argumentumból kérd be.
– Minden esetleges kivételt (főleg: IOException, vagy a bemenet beolvasáakor/konvertálásakor előforduló kivételek) kezelj le vagy kivétel specifikációval, vagy try blokkban!

Egy minta fájl felépítése az alábbi:
Pogacsa 10 150
Kave habos
Kave nem_habos

```
1
2  public interface Arlap {
3    public double mennyibeKerul();
4    public static final int CSESZEKAVE=180;
5  }
```

Arlap.java

```
1
2  public abstract class Peksutemeny implements Arlap{
3    protected double mennyiseg;
4    private double alapar;
5
6    public Peksutemeny(double mennyiseg, double alapar){
7      this.mennyiseg=mennyiseg;
8      this.alapar=alapar;
9    }
10
11   public abstract void megkostol();
12
13   public double mennyibeKerul() {
14     return alapar*mennyiseg;
15   }
16
17   public double getMennyiseg() {
```

```
18      return mennyiseg;
19    }
20
21    public void setMennyiseg(double mennyiseg) {
22      this.mennyiseg = mennyiseg;
23    }
24
25    public double getAlapar() {
26      return alapar;
27    }
28
29    public void setAlapar(double alapar) {
30      this.alapar = alapar;
31    }
32
33    @Override
34    public String toString() {
35      return mennyiseg+" db - "+mennyibeKerul()+" Ft";
36    }
37
38
39  }
```

Peksutemeny.java

```
1
2   public class Pogacsa extends Peksutemeny{
3     public Pogacsa(int mennyiseg, int alapar){
4       super(mennyiseg,alapar);
5     }
6
7     public void megkostol(){
8       this.mennyiseg=this.mennyiseg/2;
9     }
10
11    @Override
12    public String toString() {
13      return "Pogacsa " + super.toString();
14    }
15  }
```

Pogacsa.java

```
1
2  public class Kave implements Arlap {
3    private boolean habosE;
4
5    public Kave(boolean habosE){
6      this.habosE=habosE;
7    }
8
9    public double mennyibeKerul(){
10     if(habosE){
11       return CSESZEKAVE*1.5;
12     }else{
13       return CSESZEKAVE;
14     }
15   }
16
17   @Override
18   public String toString() {
19     return habosE ? "Habos kave":"Nem habos kave " + mennyibeKerul
         ()+" Ft";
20   }
21
22
23 }
```

Kave.java

```
1  import java.io.BufferedReader;
2  import java.io.BufferedWriter;
3  import java.io.File;
4  import java.io.FileNotFoundException;
5  import java.io.FileReader;
6  import java.io.FileWriter;
7  import java.io.IOException;
8  import java.util.ArrayList;
9
10 public class Pekseg {
11   public static ArrayList<Arlap> lista=new ArrayList<Arlap>();
12
13   public static void main(String[] args) {
14     try {
```

```
15        vasarlok(args[0]);
16        etelLeltar();
17      }catch(Exception ex){
18        System.out.println("Hiba tortent.");
19      }
20    }
21
22    public static void vasarlok(String path) throws IOException {
23      BufferedReader br = new BufferedReader(new FileReader(path));
24          String line = br.readLine();
25          while(line != null){
26            String[] szavak = line.split(" ");
27            if(szavak[0].equals("Pogacsa")){
28              Pogacsa p= new Pogacsa(Integer.parseInt(szavak[1]),
                    Integer.parseInt(szavak[2]));
29              lista.add(p);
30            }else if(szavak[0].equals("Kave")){
31              if(szavak[1].equals("habos")){
32                Kave k=new Kave(true);
33                lista.add(k);
34              }else if(szavak[1].equals("nem_habos")){
35                Kave k=new Kave(false);
36                lista.add(k);
37              }else {
38                System.out.println("Rossz sor");
39              }
40            }else{
41              System.out.println("Rossz sor");
42            }
43            line=br.readLine();
44          }
45
46          br.close();
47    }
48
49    public static void etelLeltar() throws IOException{
50      BufferedWriter wr = new BufferedWriter(new FileWriter("leltar.
          txt"));
51      for(Object o: lista){
52        if(o instanceof Pogacsa){
53          wr.write(((Pogacsa)o).toString()+"\n");
```

```
54          }
55        }
56        wr.close();
57      }
58  }
```

Pekseg.java

2. Oldd meg az alábbi feladatot:
   - Írj *KisGepjarmu* interfészt, ami egy *haladhatItt* metódust deklarál. A metódus egy logikai értékkel térjen vissza (haladhat -e ezen az úton a gépjármű), és egy egész számot (sebességet) kérjen paraméterként.
   - Készíts egy absztrakt *Jarmu* osztályt.
     - Egy járműnek legyen aktuális sebessége (int) és rendszáma. Az aktuális sebesség látszódjon a leszármazott osztályokban is (használd a lehető legszűkebb láthatóságot), míg a rendszám adattagot csak ebből az osztályból lehessen elérni. Írj konstruktort két paraméterrel, ami beállítja az adattagokat.
     - Készíts egy *gyorshajtottE* absztrakt metódust, ami egy sebességkorlátot (int) kér paraméternek, és logikai értékkel tér vissza attól függően , hogy az adott jármű gyorshajtott -e.
     - Készíts *toString* metódust, ami az alábbi módon alakítja szöveges formára az objektumot: "rendszam - X km/h" (ahol rendszam a jármű rendszáma, X pedig az akutális sebessége)
   - Kézsíts egy *Robogo* osztályt, ami a *Jarmu* osztályból származik és implementálja a *KisGepjarmu* interfészt.
     - A robogónak legyen egy maximális sebesség adattagja. A robogó konstruktora a rendszámot, az aktuális sebességet és a maximális sebességet kérje el paraméterül, és ez alapján hozza létre az objektumot.
     - A *gyorshajtottE* metódus nézze meg, hogy a jármű aktuális sebessége fölötte van -e a paraméterként kapott korlátnak, és ennek megfelelően térjen vissza logikai értékkel.
     - A *haladhatItt* metódust hamis értékkel térjen vissza, ha a robogó maximális sebessége nagyobb, mint a kapott paraméter, ellenkező esetben igaz legyen a visszatérés.
     - Bővítsd ki az örökölt *toString* metódust, hogy az alábbiakat adja vissza: "Robogo: rendszam - X km/h" (ahol rendszam a jármű rendszáma, X pedig az akutális sebessége) Használd fel az ősosztály toString metódusát is!
   - Kézsíts egy *AudiS8* osztályt, ami a *Jarmu* osztályból származik.
     - Az Audinak legyen egy lezerblokkolo (boolean) paramétere. Konstruktora a rendszámát, az aktuális sebességét kérje el, és hogy van -e lézerblokkolója, és ezek alapján hozza létre az objektumot.
     - A *gyorshajtottE* metódus nézze meg, hogy a jármű aktuális sebessége fölötte van -e a paraméterként kapott korlátnak, és ennek megfelelően térjen vissza logikai értékkel. Ha a jármű rendelkezik lézerblokkolóval, úgy ehelyett mindig hamissal térjen vissza.

  - Bővítsd ki az örökölt *toString* metódust, hogy az alábbiakat adja vissza: "Audi: rendszam - X km/h" (ahol rendszam a jármű rendszáma, X pedig az akutális sebessége). Használd fel az ősosztály toString metódusát is!
- Kézsíts egy *Orszagut* futtatható osztályt.
  - Az osztálynak legyen egy tárolója (tetszőleges kollekció, pl. lista), amiben Jarmu típusú objektumokat tárol.
  - Legyen tovább egy statikus *jarmuvekJonnek* metódusa. Ez egy fájl elérési útját várja paraméterül. A metódus feladata, hogy a fájlból beolvasott sorokat feldolgozza, és létrehozzon belőlük Robogo és AudiS8 objektumpéldányokat, amiket hozzáad a tárolóhoz.
  - Legyen egy statikus *kiketMertunkBe* metódus is, ami végigmegy a tárolón, és kiírja egy "buntetes.txt" fájlba a benne lévő járművek szöveges formában, és az Audi típusúakra pluszban kiírja azt is, hogy gyorshajtottak -e, míg a robogó típusúakra azt, hogy haladhatnak -e ezen az úton. Mindkét esetben 90 legyen a paraméter.
  - A main metódusban hívd meg a *jarmuvekJonnek* metódust egy parancssori argumentumból bekért elérési úttal, majd hívd meg a *kiketMertunkBe* metódust is.
  - Minden esetleges kivételt (főleg az IOException, de figyelj a bemenet feldolgozása közben tömbtúlindexelésekre és a számok átalakítása közben fellépő hibákra is) kezelj le vagy kivétel specifikációval, vagy try blokkban!

  Egy minta fájl felépítése az alábbi: (típus;rendszám;aktuális sebesség;blokkoló/max. sebesseg)
  robogo;"";40;60
  audi;AAA-111;200;true
  robogo;"";80;65
  audi;AAA-111;130;false

```
1
2  public interface KisGepjarmu {
3    public boolean haladhatItt(int sebesseg);
4  }
```

Kisgepjarmu.java

```
1
2  public abstract class Jarmu {
3    protected int sebesseg;
4    private String rendszam;
5
6    public Jarmu(int sebesseg, String rendszam){
7      this.sebesseg=sebesseg;
8      this.rendszam=rendszam;
9    }
10
```

```
11    public abstract boolean gyorshajtottE(int sebesseg);
12
13    public int getSebesseg() {
14      return sebesseg;
15    }
16
17    public void setSebesseg(int sebesseg) {
18      this.sebesseg = sebesseg;
19    }
20
21    public String getRendszam() {
22      return rendszam;
23    }
24
25    public void setRendszam(String rendszam) {
26      this.rendszam = rendszam;
27    }
28
29    @Override
30    public String toString() {
31      return rendszam+" - "+sebesseg+" km/h";
32    }
33  }
```

Jarmu.java

```
1
2   public class Robogo extends Jarmu implements KisGepjarmu{
3     private int maxsebesseg;
4
5     public Robogo(String rendszam, int aktsebesseg, int maxsebesseg)
        {
6       super(aktsebesseg, rendszam);
7       this.maxsebesseg=maxsebesseg;
8     }
9
10    public boolean gyorshajtottE(int sebesseg){
11      return this.sebesseg > sebesseg ? true : false;
12    }
13
14    public boolean haladhatItt(int sebesseg){
15      return this.maxsebesseg>sebesseg ? false : true;
```

```
16    }
17
18    public int getMaxsebesseg() {
19      return maxsebesseg;
20    }
21
22    public void setMaxsebesseg(int maxsebesseg) {
23      this.maxsebesseg = maxsebesseg;
24    }
25
26    @Override
27    public String toString() {
28      return "Robogo: "+ super.toString();
29    }
30  }
```

Robogo.java

```
1
2   public class AudiS8 extends Jarmu{
3     private boolean lezerblokkolo;
4
5     public AudiS8(String rendszam, int sebesseg, boolean
          lezerblokkolo){
6       super(sebesseg,rendszam);
7       this.lezerblokkolo=lezerblokkolo;
8     }
9
10    public boolean gyorshajtottE(int sebesseg){
11      return (!this.lezerblokkolo && sebesseg<this.sebesseg) ? true
            : false;
12    }
13
14    public boolean isLezerblokkolo() {
15      return lezerblokkolo;
16    }
17
18    public void setLezerblokkolo(boolean lezerblokkolo) {
19      this.lezerblokkolo = lezerblokkolo;
20    }
21
22    @Override
```

```
23    public String toString() {
24      return "Audi: "+ super.toString();
25    }
26
27
28  }
```

AudiS8.java

```
1   import java.io.BufferedReader;
2   import java.io.BufferedWriter;
3   import java.io.FileNotFoundException;
4   import java.io.FileReader;
5   import java.io.FileWriter;
6   import java.io.IOException;
7   import java.util.ArrayList;
8
9   public class Orszagut {
10
11    public static ArrayList<Jarmu> lista=new ArrayList<Jarmu>();
12
13    public static void jarmuvekJonnek(String path) throws
        FileNotFoundException, IOException{
14      BufferedReader br = new BufferedReader(new FileReader(path));
15          String line = br.readLine();
16          while(line != null){
17            String[] szavak = line.split(";");
18            if(szavak[0].equals("robogo")){
19              Robogo a = new Robogo(szavak[1],Integer.parseInt(
                  szavak[2]),Integer.parseInt(szavak[3]));
20              lista.add(a);
21            }else if(szavak[0].equals("audi")){
22              AudiS8 a= new AudiS8(szavak[1], Integer.parseInt(
                  szavak[2]), Boolean.parseBoolean(szavak[3]));
23              lista.add(a);
24            }else{
25              System.out.println("Hibas sor");
26            }
27            line=br.readLine();
28          }
29          br.close();
30    }
```

```
31
32    public static void kiketMertunkBe() throws IOException {
33      BufferedWriter wr = new BufferedWriter(new FileWriter("
          buntetes.txt"));
34      for(Object o: lista){
35        if(o instanceof Robogo){
36          wr.write(o.toString()+(((Robogo) o).haladhatItt(90) ? "
              haladhat\n":" nem haladhat\n"));
37        }else if(o instanceof AudiS8){
38          wr.write(o.toString()+(((AudiS8) o).gyorshajtottE(90) ? "
              gyorshajtott\n":" nem hajtott gyorsan\n"));
39        }
40      }
41      wr.close();
42    }
43
44    public static void main(String[] args) {
45      try {
46        jarmuvekJonnek(args[0]);
47        kiketMertunkBe();
48      }catch(Exception ex){
49        System.out.println("Hiba tortent.");
50      }
51    }
52  }
```

Orszagut.java

3. Oldd meg az alábbi feladatot:
   - Írj egy *Szuperhos* interfészt, ami egy *legyoziE* metódust tartalmaz. A metódus paramétere egy *Szuperhos*, és egy logikai értékkel tér vissza. Legyen egy *mekkoraAzEreje* metódusa is, ami nem kér paramétert, és a *Szuperhos* erejét fogja visszaadni.
   - Írj *Milliardos* interfészt, ami egy visszatérés nélküli kutyutKeszit() metódust tartalmaz
   - Írj egy *Bosszuallo* absztrakt osztályt, ami implementálja a *Szuperhos* interfészt.
     - Az osztály a következő private láthatóságú adattagokkal rendelkezik: egy lebegőpontos szuperero, és egy logikai vanEGyengesege.
     - Az osztály rendelkezzen paraméteres konstruktorral, ami beállítja az adattagokat. Legyen egy public *megmentiAVilagot* absztrakt metódusa, ami egy logikai értékkel tér vissza. Valósítsd meg továbbá az interfész metódusait. Az erő lekérdezésekor add vissza a *Bosszuallo* szupererejét. Egy Bosszuallo egy Bosszuallo szuperhőst akkor tud legyőzni, ha annak van gyengesége, és ereje kisebb, mint az övé. Batman-t csak akkor tudja legyőzni, ha ereje kétszer nagyobb.

  - Az osztálynak legyen továbbá getter és setter metódusa az adattagjaihoz, és legyen szöveges formára alakítható, kiírva az adattagok értékét.
- Írj egy *Vasember* osztályt, ami a *Bosszuallo* leszármazottja, és megvalósítja a *Milliardos* interfészt.
  - Az osztálynak egy default konstruktora legyen, ami beállítja a Vasember tulajdonságait. A Vasember szuperereje 150, és van gyengesége.
  - Ha a Vasember kütyüt készít, akkor szuperereje nőjön ereje egy 0-10 közötti véletlenszerű lebegőpontos számmal.
  - A Vasember akkor menti meg a világot, ha a szuperereje nagyobb, mint 1000.
  - Az osztály legyen továbbá szöveges formára alakítható. Az adattagok értékein kívül írja ki azt is, hogy a Vasemberről van szó.
- Írj egy *Batman* osztályt, ami implementálja a *Szuperhos* és *Milliardos* interfészeket.
  - Az osztálynak legyen egy lebegőpontos lelemenyesseg adattagja.
  - Az osztály rendelkezzen egy default konstruktorral, ami 100-ra állítja az adattag értékét. A metódusai az alábbiak szerint legyenek megvalósítva: Batman ereje a leleményességének kétszeresével egyezik meg, és bármilyen Szuperhőst képes legyőzni, akinek ereje kisebb, mint Batman leleményessége. Ha Batman kütyüt készít, akkor a leleményessége 50-el nő.
  - Az osztály legyen szöveges formára alakítható, ami kiírja, hogy Batmanről van szó, és megadja a leleményességét.
- Írj egy *Kepregeny* nevű futtatható osztályt. Az osztály rendelkezzen egy *szereplok* statikus függvénnyel, ami egy fájl elérési útját várja paraméterül, visszatérése pedig void. A metódus feladata, hogy a fájlból beolvasott sorokat feldolgozza, és létrehozzon belőlük *Batman*, vagy *Vasember* objektumpéldányokat, majd ezekre meghívja a kutyutKeszit metódust annyiszor, ahányszor az aktuális sor írja. Ezeket egy közös kollekcióban tárold le. Készíts továbbá egy *szuperhosok* statikus metódust, ami végigmegy a tárolóban tárolt szuperhősökön, és kiírja őket. Hívd meg a main függvényben sorban a fenti két metódust. Minden esetleges kivételt (főleg: IOException) kezelj le vagy kivétel specifikációval, vagy try blokkban!
  Egy minta fájl felépítése az alábbi:
  Vasember 5
  Batman 8

```
1
2  public interface Szuperhos {
3    public boolean legyoziE(Szuperhos o);
4
5    public double mekkoraAzEreje();
6  }
```

Szuperhos.java

```
1
2  public interface Milliardos {
3    public void kutyutKeszit();
4  }
```

Milliardos.java

```
1
2  public abstract class Bosszuallo implements Szuperhos {
3    protected double szuperero;
4    protected boolean vanEGyengesege;
5
6    public Bosszuallo(double szuperero, boolean vanEGyengesege){
7      this.szuperero=szuperero;
8      this.vanEGyengesege=vanEGyengesege;
9    }
10
11   public abstract boolean megmentiAVilagot();
12
13   public boolean legyoziE(Szuperhos o){
14     if(o instanceof Batman){
15       return true;
16     }else if (o instanceof Bosszuallo){
17       return (((Bosszuallo) o).isVanEGyengesege() && o.
             mekkoraAzEreje()<this.szuperero)? true : false;
18     }else{
19       return false;
20     }
21   }
22
23   public double mekkoraAzEreje(){
24     return this.szuperero;
25   }
26
27   public double getSzuperero() {
28     return szuperero;
29   }
30
31   public void setSzuperero(double szuperero) {
32     this.szuperero = szuperero;
33   }
```

```
34
35    public boolean isVanEGyengesege() {
36      return vanEGyengesege;
37    }
38
39    public void setVanEGyengesege(boolean vanEGyengesege) {
40      this.vanEGyengesege = vanEGyengesege;
41    }
42
43    @Override
44    public String toString() {
45      return "Bosszuallo szuperero=" + szuperero + ", vanEGyengesege
          =" + vanEGyengesege;
46    }
47  }
```

Bosszuallo.java

```
1
2   public class Vasember extends Bosszuallo implements Milliardos {
3     public Vasember(){
4       super(150,true);
5     }
6
7     public void kutyutKeszit(){
8       this.szuperero+=(Math.random()*10);
9     }
10
11    public boolean megmentiAVilagot(){
12      return this.szuperero>1000 ? true : false;
13    }
14
15    @Override
16    public String toString() {
17      return "Vasember " + super.toString();
18    }
19  }
```

Vasember.java

```
1
2   public class Batman implements Szuperhos, Milliardos{
```

```
3   private double lelemenyesseg;
4
5   public Batman(){
6     this.lelemenyesseg=100;
7   }
8
9   public boolean legyoziE(Szuperhos o){
10    return o.mekkoraAzEreje() < this.lelemenyesseg ? true : false;
11  }
12
13  public double mekkoraAzEreje(){
14    return this.lelemenyesseg*2;
15  }
16
17  public void kutyutKeszit(){
18    this.lelemenyesseg+=50;
19  }
20
21  @Override
22  public String toString() {
23    return "Batman lelemenyesseg=" + lelemenyesseg;
24  }
25
26
27 }
```

Batman.java

```
1  import java.io.BufferedReader;
2  import java.io.FileNotFoundException;
3  import java.io.FileReader;
4  import java.io.IOException;
5  import java.util.ArrayList;
6
7  public class Kepregeny {
8
9    public static ArrayList<Szuperhos> lista = new ArrayList<
       Szuperhos>();
10
11   public static void szereplok(String path) throws
       FileNotFoundException, IOException{
12     BufferedReader br = new BufferedReader(new FileReader(path));
```

```
13          String line = br.readLine();
14          while(line != null){
15            String[] szavak = line.split(" ");
16            if(szavak[0].equals("Vasember")){
17              Vasember a=new Vasember();
18              for(int i=0;i<Integer.parseInt(szavak[1]);i++){
19                a.kutyutKeszit();
20              }
21              lista.add(a);
22            }else if(szavak[0].equals("Batman")){
23              Batman a=new Batman();
24              for(int i=0;i<Integer.parseInt(szavak[1]);i++){
25                a.kutyutKeszit();
26              }
27              lista.add(a);
28            }else{
29              System.out.println("Hibas sor");
30            }
31            line=br.readLine();
32          }
33          br.close();
34    }
35
36    public static void szuperhosok(){
37      for(int i=0;i<lista.size();i++){
38        System.out.println(lista.get(i).toString());
39      }
40    }
41
42    public static void main(String[] args) {
43      try {
44        szereplok("input.txt");
45        szuperhosok();
46      }catch(Exception ex){
47        System.out.println("Hiba tortent.");
48      }
49    }
50  }
```

Kepregeny.java

4. Oldd meg az alábbi feladatot:

- Írj egy *EroErzekeny* interfészt, ami egy *legyoziE* metódust tartalmaz. A metódus paramétere egy *EroErzekeny* objektum, és egy logikai értékkel tér vissza. Legyen egy *mekkoraAzEreje* metódusa is, ami nem kér paramétert, és az *EroErzekeny* erejét fogja visszaadni.
- Írj *Sith* interfészt, ami egy visszatérés nélküli engeddElAHaragod() metódust tartalmaz
- Írj egy *Jedi* absztrakt osztályt, ami implementálja az *EroErzekeny* interfészt.
  - Az osztály a következő private láthatóságú adattagokkal rendelkezik: egy lebegőpontos ero, és egy logikai atallhatE (átállhat -e a sötét oldalra).
  - Az osztály rendelkezzen paraméteres konstruktorral, ami beállítja az adattagokat. Legyen egy public *megteremtiAzEgyensulyt* absztrakt metódusa, ami egy logikai értékkel tér vissza. Valósítsd meg továbbá az interfész metódusait. Az erő lekérdezésekor add vissza a *Jedi* erejét. Egy *Jedi* egy másik *Jedi* objektumot akkor tud legyőzni, ha az átállhat a sötét oldalra, és ereje kisebb, mint az övé. Az *Uralkodo* objektumot csak akkor tudja legyőzni, ha ereje kétszer nagyobb.
  - Az osztálynak legyen továbbá getter és setter metódusa az adattagjaihoz, és legyen szöveges formára alakítható, kiírva az adattagok értékét.
- Írj egy *AnakinSkywalker* osztályt, ami a *Jedi* leszármazottja, és megvalósítja a *Sith* interfészt.
  - Az osztálynak egy default konstruktora legyen, ami beállítja Anakin tulajdonságait. Anakin ereje 150, és átállhat a sötét oldalra.
  - Ha Anakin elengedi a haragját, akkor ereje egy 0-10 közötti véletlenszerű lebegőpontos számmal nő.
  - Anakin akkor teremti meg az egyensúlyt az erőben, ha ereje nagyobb, mint 1000.
  - Az osztály legyen továbbá szöveges formára alakítható. Az adattagok értékein kívül írja ki azt is, hogy a Anakin Skywalkerről van szó.
- Írj egy *Uralkodo* osztályt, ami implementálja az *EroErzekeny* és *Sith* interfészeket.
  - Az osztálynak legyen egy lebegőpontos gonoszsag adattagja.
  - Az osztály rendelkezzen egy default konstruktorral, ami 100-ra állítja az adattag értékét. A metódusai az alábbiak szerint legyenek megvalósítva: az Uralkodo ereje a gonoszságának kétszeresével egyezik meg, és bármilyen *EroErzekeny*-t képes legyőzni, akinek ereje kisebb, mint az Uralkodó gonoszsága. Ha az Uralkodo elengedi a haragját, akkor gonoszsága 50-el nő.
  - Az osztály legyen szöveges formára alakítható, ami kiírja, hogy az Uralkodóról van szó, és megadja a gonoszságát.
- Írj egy *StarWars* nevű futtatható osztályt. Az osztály rendelkezzen egy *szereplok* statikus függvénnyel, ami egy fájl elérési útját várja paraméterül, visszatérése pedig void. A metódus feladata, hogy a fájlból beolvasott sorokat feldolgozza, és létrehozzon belőlük *AnakinSkywalker*, vagy *Uralkodo* objektumpéldányokat, majd ezekre meghívja az engeddElAHaragod metódust annyiszor, ahányszor az aktuális sor írja. Ezeket egy közös kollekcióban tárold le. Készíts továbbá egy *sithek* statikus metódust, ami végigmegy a tárolóban tárolt objektumokon, és kiírja őket. Hívd meg a main függvényben sorban a fenti két metódust. Minden esetleges kivételt (főleg: IOException) kezelj le vagy kivétel specifikációval, vagy try blokkban!

Egy minta fájl felépítése az alábbi:
Anakin 5
Uralkodo 8

```
1
2  public interface EroErzekeny {
3    public boolean legyoziE(EroErzekeny o);
4
5    public double mekkoraAzEreje();
6  }
```

EroErzekeny.java

```
1
2  public interface Sith {
3    public void engeddElAHaragod();
4  }
```

Sith.java

```
1
2  public abstract class Jedi implements EroErzekeny{
3    protected double ero;
4    protected boolean atallhatE;
5
6    public Jedi(double ero, boolean atallhatE){
7      this.ero=ero;
8      this.atallhatE=atallhatE;
9    }
10
11   public abstract boolean megteremtiAzEgyensulyt();
12
13   public boolean legyoziE(EroErzekeny o){
14     if(o instanceof Uralkodo){
15       return o.mekkoraAzEreje()*2 < this.ero?true:false;
16     }else if(o instanceof Jedi){
17       return ((Jedi) o).isAtallhatE() && this.ero>o.mekkoraAzEreje
             () ? true : false;
18     }else{
19       return false;
20     }
21   }
```

```
22
23    public double mekkoraAzEreje(){
24      return ero;
25    }
26
27    public double getEro() {
28      return ero;
29    }
30
31    public void setEro(double ero) {
32      this.ero = ero;
33    }
34
35    public boolean isAtallhatE() {
36      return atallhatE;
37    }
38
39    public void setAtallhatE(boolean atallhatE) {
40      this.atallhatE = atallhatE;
41    }
42
43    @Override
44    public String toString() {
45      return "Jedi ero=" + ero + ", atallhatE=" + atallhatE;
46    }
47
48
49  }
```

Jedi.java

```
1
2   public class AnakinSkywalker extends Jedi implements Sith {
3     public AnakinSkywalker(){
4       super(150,true);
5     }
6
7     public void engeddElAHaragod(){
8       this.ero+=(Math.random()*10);
9     }
10
11    public boolean megteremtiAzEgyensulyt(){
```

```
12      return this.ero > 1000 ? true : false;
13    }
14
15    @Override
16    public String toString() {
17      return "AnakinSkywalker " + super.toString();
18    }
19  }
```

AnakinSkywalker.java

```
1
2   public class Uralkodo implements EroErzekeny, Sith{
3     private double gonoszsag;
4
5     public Uralkodo(){
6       this.gonoszsag=100;
7     }
8
9     public boolean legyoziE(EroErzekeny o){
10      return o.mekkoraAzEreje()<this.gonoszsag ? true : false;
11    }
12
13    public double mekkoraAzEreje(){
14      return this.gonoszsag*2;
15    }
16
17    public void engeddElAHaragod(){
18      this.gonoszsag+=50;
19    }
20
21    public double getGonoszsag() {
22      return gonoszsag;
23    }
24
25    public void setGonoszsag(double gonoszsag) {
26      this.gonoszsag = gonoszsag;
27    }
28
29    @Override
30    public String toString() {
31      return "Uralkodo gonoszsag=" + gonoszsag;
```

```
  }
}
```

Uralkodo.java

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;

public class StarWars {

  public static ArrayList<EroErzekeny> lista=new ArrayList<
     EroErzekeny>();

  public static void szereplok(String path) throws IOException,
     FileNotFoundException{
    BufferedReader br = new BufferedReader(new FileReader(path));
        String line = br.readLine();
        while(line != null){
          String[] szavak = line.split(" ");
          if(szavak[0].equals("Anakin")){
            AnakinSkywalker a=new AnakinSkywalker();
            for(int i=0;i<Integer.parseInt(szavak[1]);i++){
              a.engeddElAHaragod();
            }
            lista.add(a);
          }else if(szavak[0].equals("Uralkodo")){
            Uralkodo a=new Uralkodo();
            for(int i=0;i<Integer.parseInt(szavak[1]);i++){
              a.engeddElAHaragod();
            }
            lista.add(a);
          }else{
            System.out.println("Rossz sor");
          }
          line=br.readLine();
        }
        br.close();
  }

```

```
36    public static void sithek(){
37      for(int i=0;i<lista.size();i++){
38        System.out.println(lista.get(i).toString());
39      }
40    }
41
42    public static void main(String[] args) {
43      try {
44        szereplok("input.txt");
45        sithek();
46      }catch(Exception ex){
47        System.out.println("Hiba tortent.");
48      }
49    }
50  }
```

StarWars.java

5. Oldd meg az alábbi feladatot:
   - Írj egy *Urhajo* interfészt, ami egy *legyorsuljaE* metódust tartalmaz. A metódus paramétere egy *Urhajo* objektum, és egy logikai értékkel tér vissza. Legyen egy *milyenGyors* metódusa is, ami nem kér paramétert, és az *Urhajo* gyorsaságát fogja visszaadni.
   - Írj *Hiperhajtomu* interfészt, ami egy visszatérés nélküli hiperUgras() metódust tartalmaz
   - Írj egy *LazadoGep* absztrakt osztályt, ami implementálja az *Urhajo* interfészt.
     - Az osztály a következő private láthatóságú adattagokkal rendelkezik: egy lebegőpontos sebesség, és egy logikai meghibasodhatE.
     - Az osztály rendelkezzen paraméteres konstruktorral, ami beállítja az adattagokat. Legyen egy public *elkapjaAVonosugar* absztrakt metódusa, ami egy logikai értékkel tér vissza. Valósítsd meg továbbá az interfész metódusait. A gyorsaság lekérdezésekor add vissza a *LazadoGep* sebességét. Egy *LazadoGep* egy másik *LazadoGep* objektumot akkor tud legyorsulni, ha az meghibásodhat, és a gyorsasága kisebb, mint az övé. A *MilleniumFalcon* objektumot csak akkor tudja legyőzni, ha gyorsasága kétszer nagyobb.
     - Az osztálynak legyen továbbá getter és setter metódusa az adattagjaihoz, és legyen szöveges formára alakítható, kiírva az adattagok értékét.
   - Írj egy *XWing* osztályt, ami a *LazadoGep* leszármazottja, és megvalósítja az *Hiperhajtomu* interfészt.
     - Az osztálynak egy default konstruktora legyen, ami beállítja az X-Wing tulajdonságait. Az X-Wing sebessége 150, és meghibásodhat.
     - Ha az X-Wing hiperugrást végez, akkor sebessége egy 0-100 közötti véletlenszerű lebegőpontos számmal nő.
     - Az X-Wing et akkor kapja el a vonósugár, ha meghibásodhat, és sebessége kisebb, mint 10000.

  - Az osztály legyen továbbá szöveges formára alakítható. Az adattagok értékein kívül írja ki azt is, hogy egy X-Wingről van szó.
- Írj egy *MilleniumFalcon* osztályt, ami implementálja az *Urhajo* és *Hiperhajtomu* interfészeket.
  - Az osztálynak legyen egy lebegőpontos tapasztalat adattagja.
  - Az osztály rendelkezzen egy default konstruktorral, ami 100-ra állítja az adattag értékét. A metódusai az alábbiak szerint legyenek megvalósítva: a Millenium Falcon gyorsasága a tapasztalatának kétszeresével egyezik meg, és bármilyen *Urhajo*-t képes legyorsulni, akinek gyorsasága kisebb, mint a Falcon gyorsasága. Ha a Millenium Falcon hiperugrást végez, akkor tapasztalata 500-al nő.
  - Az osztály legyen szöveges formára alakítható, ami kiírja, hogy a Millenium Falconról van szó, és megadja a tapasztalatát.
- Írj egy *StarWars* nevű futtatható osztályt. Az osztály rendelkezzen egy *urhajok* statikus függvénnyel, ami egy fájl elérési útját várja paraméterül, visszatérése pedig void. A metódus feladata, hogy a fájlból beolvasott sorokat feldolgozza, és létrehozzon belőlük *XWing*, vagy *MilleniumFalcon* objektumpéldányokat, majd ezekre meghívja a hiperUgras metódust annyiszor, ahányszor az aktuális sor írja. Ezeket egy közös kollekcióban tárold le. Készíts továbbá egy *hangar* statikus metódust, ami végigmegy a tárolóban tárolt objektumokon, és kiírja őket. Hívd meg a main függvényben sorban a fenti két metódust. Minden esetleges kivételt (főleg: IOException) kezelj le vagy kivétel specifikációval, vagy try blokkban!
  Egy minta fájl felépítése az alábbi:
  XWing 4
  MilleniumFalcon 18

```
1
2  public interface Urhajo {
3    public boolean legyorsuljaE(Urhajo o);
4
5    public double milyenGyors();
6  }
```

Urhajo.java

```
1
2  public interface Hiperhajtomu {
3    public void hiperUgras();
4  }
```

Hiperhajtomu.java

```
1
2  public abstract class LazadoGep implements Urhajo{
```

```
3   protected double sebesseg;
4   protected boolean meghibasodhatE;
5
6   public LazadoGep(double sebesseg, boolean meghibasodhatE){
7     this.sebesseg=sebesseg;
8     this.meghibasodhatE=meghibasodhatE;
9   }
10
11  public abstract boolean elkapjaAVonosugar();
12
13  public boolean legyorsuljaE(Urhajo o){
14    if(o instanceof MilleniumFalcon){
15      return (((MilleniumFalcon)o).milyenGyors() < 2*this.sebesseg
          ) ? true: false;
16    }else{
17      return (((LazadoGep)o).isMeghibasodhatE() && o.milyenGyors()
          <this.sebesseg) ? true : false;
18    }
19  }
20
21  public double getSebesseg() {
22    return sebesseg;
23  }
24
25  public void setSebesseg(double sebesseg) {
26    this.sebesseg = sebesseg;
27  }
28
29  public boolean isMeghibasodhatE() {
30    return meghibasodhatE;
31  }
32
33  public void setMeghibasodhatE(boolean meghibasodhatE) {
34    this.meghibasodhatE = meghibasodhatE;
35  }
36
37  public double milyenGyors(){
38    return this.sebesseg;
39  }
40
41  @Override
```

```
42   public String toString() {
43     return "LazadoGep sebesseg=" + sebesseg + ", meghibasodhatE="
          + meghibasodhatE;
44   }
45 }
```

LazadoGep.java

```
1
2  public class XWing extends LazadoGep implements Hiperhajtomu {
3    public XWing(){
4      super(150,true);
5    }
6
7    public void hiperUgras(){
8      this.sebesseg+=Math.random()*100;
9    }
10
11   public boolean elkapjaAVonosugar(){
12     return (this.meghibasodhatE && sebesseg<10000) ? true : false;
13   }
14
15   @Override
16   public String toString() {
17     return "XWing "+super.toString();
18   }
19 }
```

XWing.java

```
1
2  public class MilleniumFalcon implements Urhajo, Hiperhajtomu {
3    private double tapasztalat;
4
5    public MilleniumFalcon(){
6      this.tapasztalat=100;
7    }
8
9    public boolean legyorsuljaE(Urhajo o){
10     return o.milyenGyors() < this.milyenGyors() ? true : false;
11   }
12
```

```
13    public double milyenGyors(){
14      return tapasztalat*2;
15    }
16
17    public void hiperUgras(){
18      this.tapasztalat+=500;
19    }
20
21    @Override
22    public String toString() {
23      return "MilleniumFalcon tapasztalat=" + tapasztalat;
24    }
25  }
```

MilleniumFalcon.java

```
1   import java.io.BufferedReader;
2   import java.io.FileNotFoundException;
3   import java.io.FileReader;
4   import java.io.IOException;
5   import java.util.ArrayList;
6
7   public class StarWars {
8
9     public static ArrayList<Hiperhajtomu> lista = new ArrayList<
        Hiperhajtomu>();
10
11    public static void urhajok(String path) throws
        FileNotFoundException, IOException {
12      BufferedReader br = new BufferedReader(new FileReader(path));
13          String line = br.readLine();
14          while(line != null){
15            String[] szavak = line.split(" ");
16            if(szavak[0].equals("XWing")){
17              XWing a=new XWing();
18              for(int i=0;i<Integer.parseInt(szavak[1]);i++){
19                a.hiperUgras();
20              }
21              lista.add(a);
22            }else if(szavak[0].equals("MilleniumFalcon")){
23              MilleniumFalcon a=new MilleniumFalcon();
24              for(int i=0;i<Integer.parseInt(szavak[1]);i++){
```

```
25                a.hiperUgras();
26              }
27              lista.add(a);
28            }else{
29              System.out.println("Rossz sor");
30            }
31            line=br.readLine();
32          }
33          br.close();
34    }
35
36    public static void hangar(){
37      for(int i=0;i<lista.size();i++){
38        System.out.println(lista.get(i).toString());
39      }
40    }
41
42    public static void main(String[] args) {
43      try {
44        urhajok("input.txt");
45        hangar();
46      }catch(Exception ex){
47        System.out.println("Hiba tortent.");
48      }
49    }
50  }
```

StarWars.java

6. Oldd meg az alábbi feladatot:
   - Írj *Akciozhato* interfészt, ami egy *akciosAr* metódust deklarál. A metódus egy egész számmal térjen vissza (egy dolog akciós ára), és ne kérjen paramétert.
   - Készíts egy absztrakt *Termek* osztályt.
     - Egy terméknek legyen neve, és egységára (int). A név adattagot csak ebből az osztályból lehessen elérni, az egységár látszódjon a leszármazott osztályokban is (használd a lehető legszűkebb láthatóságot). Írj konstruktort két paraméterrel, ami beállítja az adattagokat.
     - Készíts egy *mennyibeKerul* absztrakt metódust, ami nem kér paramétert, és a termék tényleges árával (int) tér vissza.
     - Készíts *toString* metódust, ami az alábbi módon alakítja szöveges formára az objektumot: "nev - X Ft" (ahol nev a termék neve, X pedig a tényleges ára)
   - Kézsíts egy *Salata* osztályt, ami a *Termek* osztályból származik.

  - A salátának legyen egy darab paramétere (int), ami megadja, hogy mennyit veszünk belőle. Konstruktora a darabszámot és az egységárat kérje el paraméterül, és ez alapján hozza létre az objektumot (a neve legyen "salata").
  - A *mennyibekerul* metódus a darabszám és az egységár szorzataként határozza meg a saláta árát.
  - Bővítsd ki az örökölt *toString* metódust, hogy az alábbiakat adja vissza: "Y db salata - X Ft" (ahol Y a darabszám, X a tényleges ár). Használd fel az ősosztály toString metódusát is!
- Kézsíts egy *RohadtParadicsom* osztályt, ami a *Termek* osztályból származik, és implementálja az *Akciozhato* interfészt.
  - A rohadt paradicsomnak legyen egy tomeg paramétere (double), ami megadja, hogy mennyit veszünk belőle. Konstruktora a tömegét és az egységárat kérje el paraméterül, és ez alapján hozza létre az objektumot (a neve legyen "rohadt paradicsom").
  - A *mennyibekerul* metódus a tömeg és az egységár szorzataként határozza meg a paradicsom árát. Figyelj a kerekítésre!
  - A paradicsom akciós ára a termék tényleges árának 80%-a legyen (ez is int, itt is figyelj a kerekítésre!)
  - Bővítsd ki az örökölt *toString* metódust, hogy az alábbiakat adja vissza: "Y kg rohadt paradicsom - X Ft" (ahol Y a darabszám, X a tényleges ár). Használd fel az ősosztály toString metódusát is!
- Kézsíts egy *Vegyesbolt* futtatható osztályt.
  - Az osztálynak legyen egy tárolója (tetszőleges kollekció, pl. lista), amiben Termek típusú objektumokat tárol.
  - Legyen tovább egy statikus *bevasarlok* metódusa. Ez egy fájl elérési útját várja paraméterül. A metódus feladata, hogy a fájlból beolvasott sorokat feldolgozza, és létrehozzon belőlük Salata és RohadtParadicsom objektumpéldányokat, amiket hozzáad a tárolóhoz.
  - Legyen egy statikus *mivanakosaramban* metódus is, ami végigmegy a tárolón, és kiírja egy "kosar.txt" fájlba a benne lévő termékeket szöveges formában. Ha valamelyik termék akciózható típusú, úgy annak írja ki az akciós árát is a szöveges információ mellé.
  - A main metódusban hívd meg a *bevasarlok* metódust egy parancssori argumentumból bekért elérési úttal, majd hívd meg a *mivanakosaramban* metódust is.
  - Minden esetleges kivételt (főleg az IOException, de figyelj a bemenet feldolgozása közben tömbtúlindexelésekre és a számok átalakítása közben fellépő hibákra is) kezelj le vagy kivétel specifikációval, vagy try blokkban!

  Egy minta fájl felépítése az alábbi: (név;egységár;egység)
  salata;400;2
  paradicsom;540;0.5
  salata;349;5

```
1
```

```
2  public interface Akciozhato {
3    public int akciosAr();
4  }
```

Akciozhato.java

```
1
2  public abstract class Termek {
3    private String nev;
4    protected int egysegar;
5
6    public Termek(String nev, int egysegar){
7      this.nev=nev;
8      this.egysegar=egysegar;
9    }
10
11   public abstract int mennyibeKerul();
12
13   @Override
14   public String toString() {
15     return nev + " - " + mennyibeKerul()+" Ft";
16   }
17 }
```

Termek.java

```
1
2  public class Salata extends Termek{
3    private int vettdb;
4
5    public Salata(int vettdb,int egysegar){
6      super("salata",egysegar);
7      this.vettdb=vettdb;
8    }
9
10   public int mennyibeKerul(){
11     return this.vettdb*this.egysegar;
12   }
13
14   public int getVettdb() {
15     return vettdb;
16   }
```

```
17
18    public void setVettdb(int vettdb) {
19      this.vettdb = vettdb;
20    }
21
22    @Override
23    public String toString() {
24      return this.vettdb+" db "+super.toString();
25    }
26  }
```

Salata.java

```
1
2   public class RohadtParadicsom extends Termek implements Akciozhato
        {
3     private double tomeg;
4
5     public RohadtParadicsom(double tomeg, int egysegar){
6       super("rohadt paradicsom",egysegar);
7       this.tomeg=tomeg;
8     }
9
10    public int mennyibeKerul(){
11      return (int)Math.round(this.tomeg*this.egysegar);
12    }
13
14    public int akciosAr(){
15      return (int)Math.round(mennyibeKerul()*0.8);
16    }
17
18    @Override
19    public String toString() {
20      return this.tomeg+" kg "+super.toString();
21    }
22  }
```

RohadtParadicsom.java

```
1   import java.io.BufferedReader;
2   import java.io.FileNotFoundException;
3   import java.io.FileReader;
```

```
4  import java.io.IOException;
5  import java.util.ArrayList;
6
7  public class Vegyesbolt {
8
9    public static ArrayList<Termek> lista = new ArrayList<Termek>();
10
11   public static void bevasarlok(String path) throws
       FileNotFoundException, IOException{
12     BufferedReader br = new BufferedReader(new FileReader(path));
13         String line = br.readLine();
14         while(line != null){
15           String[] szavak = line.split(";");
16           if(szavak[0].equals("salata")){
17             Salata a=new Salata(Integer.parseInt(szavak[1]),
                 Integer.parseInt(szavak[2]));
18             lista.add(a);
19           }else if(szavak[0].equals("paradicsom")){
20             RohadtParadicsom a=new RohadtParadicsom(Double.
                 parseDouble(szavak[2]),Integer.parseInt(szavak[1]))
                 ;
21             lista.add(a);
22           }else{
23             System.out.println("Rossz sor");
24           }
25           line=br.readLine();
26         }
27         br.close();
28   }
29
30   public static void mivanakosaramban(){
31     for(Object o:lista){
32       if(o instanceof RohadtParadicsom){
33         System.out.println(((RohadtParadicsom) o).toString()+" az
             akcios ara:"+((RohadtParadicsom) o).akciosAr());
34       }else{
35         System.out.println(((Salata) o).toString());
36       }
37     }
38   }
39
```

```
40    public static void main(String[] args) {
41      try {
42        bevasarlok(args[0]);
43        mivanakosaramban();
44      }catch(Exception ex){
45        System.out.println("Hiba tortent.");
46      }
47    }
48  }
```

Vegyesbolt.java

7. Oldd meg az alábbi feladatot:
   - Írj egy *Ingatlan* interfészt, ami egy paraméter nélküli *osszesKoltseg* metódust tartalmaz. A metódus az ingatlan költségét adja majd vissza.
   - Írj egy *Berelheto* interfészt, aminek három metódusa van. Az egyik a *mennyibeKerul*, ami egy egész számot (hónapok száma) kap paraméterül, és egy számmal (a bérleti díjjal) tér majd vissza. A másik a paraméter nélküli *lefoglaltE*, ami egy logikai értékkel tér majd vissza, hogy lefoglalták -e már az adott dolgot. A harmadik *lefoglal*, ami egy számot (lefoglalni kívánt hónapok száma) vár paraméterül, és logikai értékkel tér vissza attól függően, hogy sikeres volt -e a foglalás.
   - Írj egy *Lakas* absztrakt osztályt, ami implementálja az *Ingatlan* interfészt.
     - Az osztály a következő adattagokkal rendelkezik: terület (terulet), szobák száma (szobaSzam), lakók száma (lakok), négyzetméter ár (nmAr). Az adattagok legyenek láthatóak a leszármazott osztályokban is! Az osztály rendelkezzen paraméteres konstruktorral, ami beállítja az adattagok értékét.
     - Készíts egy egész paramétert váró *bekoltozik* absztrakt metódust, ami embereket költöztet a lakásba. A metódus logikai értékkel tér vissza attól függően, hogy sikeres volt -e a beköltözés.
     - Valósítsd meg az interfész metódusát, ami a terület és a négyzetméterenkénti ár alapján visszaadja a lakás teljes költségét.
     - Készíts egy paraméter nélküli *lakokSzama* metódust, ami visszadja a lakásban lakók számát.
     - Készíts egy toString metódust, ami Stringgé alakítja az osztály tulajdonságait.
   - Írj egy *Alberlet* osztályt, ami a *Lakas*-ból származik és implementálja a *Berelheto* interfészt.
     - Az örökölteken kívül egy foglalt hónapok szama (honapSzam) adattaggal rendelkezik, ami megadja, hogy hány hónapra van lefoglalva az albérlet. Készíts paraméteres konstruktort, ami az ősosztály hasonló konstruktorát használva beállítja az adattagok értékét. A foglalt hónapok és a lakók száma kezdeti értéke 0 legyen!
     - Implementáld az interfész első metódusát. Az albérlet egy főre jutó havi költsége a lakás összes költségének és a lakók számának hányadosa, a metódus ez alapján adja

vissza a kívánt időszak költségét. Ha nincs a lakásban lakó, úgy -1 legyen a visszatérési érték!

- Implementáld az interfész második metódusát. Az albérlet akkor nincs lefoglalva, ha a foglalt hónapok száma 0.
- Implementáld az interfész harmadik metódusát. A lakás foglaltságától függően, módosítsd a foglalt hónapok számát (ha még nem volt lefoglalva), és térj vissza megfelelő értékkel.
- Implementáld az ősosztály absztrakt metódusát az alábbiak szerint: az albérletben egy szobában maximum 8 fő lakhat, és egy főre minimum 2 m^2 területnek kell jutnia. A beköltözés, és a metódus visszatérése ettől a feltételtől függjön. Ha a feltétel teljesül, úgy módosítsd a lakók számát is.
- Írd felül a toString metódust, hogy már lefoglalt hónapok száma is szerepeljen benne. A metóduson belül használd az ősosztály toString-jét is.

- Írj egy *CsaladiApartman* osztályt, ami a *Lakas*-ból származik.
  - Az örökölteken kívül egy gyerekekSzama adattaggal rendelkezik, ami megadja, hogy az összes lakóból mennyi a gyerek. Készíts paraméteres konstruktort, ami az ősosztály hasonló konstruktorát használva beállítja az adattagok értékét. A lakók és gyerekek száma kezdeti értéke is 0 legyen.
  - Készíts egy paraméter nélküli *gyerekSzuletik* metódust. A metódus ellenőrizze le, hogy van -e két felnőtt lakója az apartmannak, és ha igen, úgy növelje a lakók és gyerekek számát is 1-el. Térjen vissza logikai értékkel attól függően, hogy megszületett -e a gyerek.
  - Implementáld az ősosztály absztrakt metódusát az alábbiak szerint. az apartman egy szobájában maximum 2 fő lakhat, és egy főre minimum 10 m^2 területnek kell jutnia. Gyerekek ebből a szempontból fél főnek számítanak csak. A beköltöző lakók mind felnőttek. A metódus visszatérése a fenti feltételtől függjön. Ha a feltétel teljesül, úgy módosítsd a lakók számát is.
  - Írd felül a toString metódust, hogy a gyereke száma is szerepeljen benne. A metóduson belül használd az ősosztály toString-jét is.
- Írj egy *Garazs* osztályt, ami implementálja mindkét fenti interfészt.
  - Az osztály a következő adattagokkal rendelkezik: terület (terulet), négyzetméter ár (nmAr), fűtött -e (futottE), foglalt hónapok szama (honapSzam), áll -e benne autó (auto). Készíts paraméteres konstruktort, ami beállítja az adattagok értékét. A garázsban kezdetben ne álljon autó, és a foglalt hónapok száma 0 legyen!
  - Implementáld az *Ingatlan* interfész metódusát! A garázs összes költségét a terület és a négyzetméterenkénti ár alapján kapot.
  - Implementáld a *Berelheto* interfész metódusait! A garázs havi költsége a terület és ár alapján számolható, ezt még 1,5-el kell szorozni, ha fűtött is. Ez alapján visszadható a kívánt időszak költsége. A garázs akkor foglalt, ha a foglalt hónapok száma nagyobb, mint 0, vagy áll benne autó. A garázs foglaltságától függően módosítsd a foglalt hónapok számát (ha még nem volt lefoglalva), és térj vissza megfelelő értékkel.
  - Készíts egy paraméter nélküli autoKiBeAll metódust. Ha már áll autó a garázsban,

úgy az álljon ki, ha pedig nem áll, úgy álljon be. A metódus ne térjen vissza semmmivel.
  - Készíts egy toString metódust, ami Stringgé alakítja az osztály tulajdonságait.
- Írj egy *Tarsashaz* nevű osztályt.
  - Az osztálynak egy adattagja legyen: egy kollekció (pl. lista), ami képes Lakas és Garazs objektumokat is tárolni, valamint két egész szám, hogy maximálisan hány lakás és garázs lehet a társasházban. Írj konstruktort, ami a lakások és garázsok maximális számát kapja paraméternek, és beállítja a megfelelő adattagoat. Kezdetben a nincs lakás és garázs a házban.
  - Írj két metódust *lakasHozzaad* és *garazsHozzaad* néven. Mindegyik metódussal a neki megfelelő objektumot helyezhetjük majd el a házban, így paraméternek mindegyik a megfelelő típust várja. A metódusok adják hozzá a kollekcióhoz a paraméterként kapott objektumot, ha még nem értük el az adott típusból a maximális mennyiséget. Visszatérési értékük logikai legyen attól függően, hogy sikerült -e a hozzáadás, vagy sem.
  - Írj egy paraméter nélküli osszesLako metódust, ami visszadja, hogy a házban található lakásokban összesen hány lakó van.
  - Írj egy paraméter nélküli ingatlanErtek metódust. A metódus számolja végig a házban lévő és használatban lévő lakások és garázsok értékét, és ezek összegével térjen vissza. Lakások esetén azok vannak használatban, amikben legalább 1 lakó lakik, míg garázs esetén az, ami le van foglalva.
- Írj egy *Hazmester* nevű futtatható osztályt. Az osztály rendelkezzen egy *karbantart* statikus függvénnyel, ami egy fájl elérési útját várja paraméterül, visszatérési értéke pedig void. A metódus példányosítson egy Tarsashaz-at. A metódus feladata ezután, hogy a fájlból beolvasott sorokat (ingatlanokat) feldolgozza, és létrehozzon belőlük Alberlet, CsaladiApartman és Garazs objektumokat. Ezeket az obejtumokat a megfelelő függvény használatával adja is hozzá a létrehozott társasházhoz. Miután végzett a fájl feldolgozásával, írja ki konzolra, hogy mennyi a társasház összes értéke. Minden esetleges kivételt (főleg FileNotFoundException és IOException) kezelj le kivételspecifikációval, vagy try blokkba. Egy minta fájl felépítése az alábbi:
  Alberlet 50.2 5 13000
  CsaladiApartman 62.8 2 40000
  Garazs 10.3 5000 futott

```
1
2 public interface Ingatlan {
3   public double koltseg();
4 }
```

Ingatlan.java

```
1
2 public interface Berelheto {
```

```
  public double mennyibeKerul(int honapokszama);

  public boolean lefoglaltE();

  public boolean lefoglal(int honapokszama);
}
```

Berelheto.java

```

public abstract class Lakas implements Ingatlan{
  protected double terulet;
  protected int szobaSzam;
  protected int lakok;
  protected int nmAr;

  public Lakas(double terulet, int szobaSzam, int lakok, int nmAr)
     {
    this.terulet=terulet;
    this.szobaSzam=szobaSzam;
    this.lakok=lakok;
    this.nmAr=nmAr;
  }

  public abstract boolean bekoltozik(int emberek);

  public double koltseg(){
    return this.terulet*this.nmAr;
  }

  public int lakokSzama(){
    return this.lakok;
  }

  @Override
  public String toString() {
    return "Lakas, Terulet:"+terulet+", szobaszam:"+szobaSzam+",
       lakokszama:"+lakok+", nmAr:"+nmAr;
  }
}
```

Lakas.java

```
1
2  public class Alberlet extends Lakas implements Berelheto{
3    int honapSzam;
4
5    public Alberlet(double terulet, int szobaSzam, int nmAr){
6      super(terulet, szobaSzam, 0, nmAr);
7      this.honapSzam=0;
8    }
9
10   public double mennyibeKerul(int honapokszama){
11     return this.lakok == 0 ? -1 : (double)koltseg()/lakok;
12   }
13
14   public boolean lefoglaltE(){
15     return this.honapSzam == 0 ? false : true;
16   }
17
18   public boolean lefoglal(int honapokszama){
19     if(lefoglaltE()){
20       return false;
21     }else{
22       this.honapSzam=honapokszama;
23       return true;
24     }
25   }
26
27   public boolean bekoltozik(int emberek){
28     if((double)this.szobaSzam/(double)(this.lakok+emberek)<=8.0 &&
           (double)terulet/(double)(this.lakok+emberek)>=2.0){
29       this.lakok+=emberek;
30       return true;
31     }else{
32       return false;
33     }
34   }
35
36   @Override
37   public String toString() {
38     return super.toString()+", honapokszama: "+honapSzam;
39   }
```

```
40
41
42 }
```

Alberlet.java

```
1
2  public class CsaladiApartman extends Lakas{
3    int gyerekekSzama;
4
5    public CsaladiApartman(double terulet, int szobaSzam, int nmAr){
6      super(terulet, szobaSzam, 0, nmAr);
7      this.gyerekekSzama=0;
8    }
9
10   public boolean gyerekSzuletik(){
11     if(lakok>=2){
12       this.gyerekekSzama++;
13       return true;
14     }else{
15       return false;
16     }
17   }
18
19   public boolean bekoltozik(int emberek){
20     double szemelyekszama=this.lakok+(0.5*(double)gyerekekSzama);
21     if(this.szobaSzam/szemelyekszama<=2.0 && this.terulet/
          szemelyekszama>=2.0){
22       this.lakok+=emberek;
23       return true;
24     }else{
25       return false;
26     }
27   }
28
29   @Override
30   public String toString() {
31     return super.toString()+", gyerekekszama: "+this.gyerekekSzama
          ;
32   }
33 }
```

CsaladiApartman.java

```
1
2  public class Garazs implements Ingatlan, Berelheto{
3    double terulet;
4    int nmAr;
5    boolean futottE;
6    int honapSzam;
7    boolean auto;
8
9    public Garazs(double terulet, int nmAr, boolean futottE){
10     this.terulet=terulet;
11     this.nmAr=nmAr;
12     this.futottE=futottE;
13     this.honapSzam=0;
14     this.auto=false;
15   }
16
17   public double koltseg(){
18     return this.terulet*this.nmAr;
19   }
20
21   public double mennyibeKerul(int honapokszama){
22     return this.futottE ? this.terulet*this.nmAr*1.5 : this.
         terulet*this.nmAr;
23   }
24
25   public boolean lefoglaltE(){
26     return this.honapSzam == 0 && !this.auto ? false : true;
27   }
28
29   public boolean lefoglal(int honapokszama){
30     if(lefoglaltE()){
31       return false;
32     }else{
33       this.honapSzam=honapokszama;
34       return true;
35     }
36   }
37
```

```
38    public void autoKiBeAll(){
39      this.auto=!this.auto;
40    }
41
42    @Override
43    public String toString() {
44      return "Garazs terulet:" + terulet + ", nmAr:" + nmAr + ",
           futottE:" + futottE + ", honapSzam:" + honapSzam
45        + ", auto:" + auto + "]";
46    }
47
48
49  }
```

Garazs.java

```
1   import java.util.ArrayList;
2
3   public class Tarsashaz {
4     ArrayList<Ingatlan> lista;
5     int maxgarazs;
6     int maxlakas;
7
8     public Tarsashaz(int maxgarazs, int maxlakas){
9       this.lista=new ArrayList<Ingatlan>();
10      this.maxgarazs=maxgarazs;
11      this.maxlakas=maxlakas;
12    }
13
14    public boolean lakasHozzaad(Lakas l){
15      int lakasokszama=0;
16      for(Object o: this.lista){
17        if(o instanceof Lakas){
18          lakasokszama++;
19        }
20      }
21      if(lakasokszama<=this.maxlakas){
22        lista.add(l);
23        return true;
24      }else{
25        return false;
26      }
```

```
27    }
28
29    public boolean GarazsHozzaad(Garazs g){
30      int garazsokszama=0;
31      for(Object o: this.lista){
32        if(o instanceof Garazs){
33          garazsokszama++;
34        }
35      }
36      if(garazsokszama<=this.maxgarazs){
37        lista.add(g);
38        return true;
39      }else{
40        return false;
41      }
42    }
43
44    public int osszesLako(){
45      int lakok=0;
46      for(Object o: this.lista){
47        if(o instanceof Lakas){
48          lakok+=((Lakas) o).lakokSzama();
49        }
50      }
51      return lakok;
52    }
53
54    public int ingatlanErtek(){
55      int ertek=0;
56      for(Object o:this.lista){
57        if(o instanceof Lakas){
58          if(((Lakas) o).lakokSzama()>=1){
59            ertek+=((Lakas) o).koltseg();
60          }
61        }else if(o instanceof Garazs){
62          if(((Garazs) o).lefoglaltE()){
63            ertek+=((Garazs) o).koltseg();
64          }
65        }
66      }
67      return ertek;
```

```
68    }
69 }
```

Tarsashaz.java

```
1  import java.io.BufferedReader;
2  import java.io.FileNotFoundException;
3  import java.io.FileReader;
4  import java.io.IOException;
5
6  public class Hazmester {
7
8    public static void karbantart(String path) throws IOException,
       FileNotFoundException{
9      Tarsashaz t=new Tarsashaz(100,100);
10
11     BufferedReader br = new BufferedReader(new FileReader(path));
12         String line = br.readLine();
13         while(line != null){
14           String[] szavak = line.split(" ");
15           if(szavak[0].equals("Alberlet")){
16             Alberlet a = new Alberlet(Double.parseDouble(szavak
                 [1]),Integer.parseInt(szavak[2]),Integer.parseInt(
                 szavak[3]));
17             t.lakasHozzaad(a);
18           }else if(szavak[0].equals("CsaladiApartman")){
19             CsaladiApartman a = new CsaladiApartman(Double.
                 parseDouble(szavak[1]),Integer.parseInt(szavak[2]),
                 Integer.parseInt(szavak[3]));
20             t.lakasHozzaad(a);
21           }else if(szavak[0].equals("Garazs")){
22             if(szavak[3].equals("futott")){
23               Garazs a = new Garazs(Double.parseDouble(szavak[1]),
                   Integer.parseInt(szavak[2]), true);
24               t.GarazsHozzaad(a);
25             }else{
26               Garazs a = new Garazs(Double.parseDouble(szavak[1]),
                   Integer.parseInt(szavak[2]), false);
27               t.GarazsHozzaad(a);
28             }
29           }
30           line=br.readLine();
```

```
31          }
32          br.close();
33
34          System.out.println("A tarsashaz erteke:"+t.ingatlanErtek()
               );
35      }
36
37      public static void main(String[] args) throws IOException,
           FileNotFoundException {
38
39        try {
40          karbantart("input.txt");
41        }catch(Exception ex){
42          System.out.println("Hiba tortent.");
43        }
44
45      }
46  }
```

Hazmester.java

8. Oldd meg az alábbi feladatot:
   - Írj Arak interfészt, ami két konstanst tartalmaz: EGYAGYAS és KETAGYAS, amiknek értékei 8000 és 12000.
   - Írj egy Kedvezmenyes interészt. Legyen az interfésznek egy kedvezmenytKer absztrakt metódusa, aminek visszatérési típusa void, paramétere nincs.
   - Készíts egy absztrakt Szoba osztályt, ami implementálja az Arak interfeszt.
     - A következő adattagokkal rendelkezik: berletiDij, fekvoHely, lakok.
     - Írj paraméteres konstruktort, ami a lakók számát kapja paraméternek, és azok alapján beállítja az adattagokat. Mindig annyi fekvőhely legyen, ahány lakó. A berletiDij értéke a konstansoktól függ, 2-nél több fekvőhely esetén minden plusz személyre az egyágyas felárral nő.
     - A szobának legyen toString metódusa, ami kiírja az adattagok értékeit.
     - Legyen az osztálynak egy kikoltozik metódusa, ami egy egész számot vár paraméternek, és ennyivel csökkenti a lakók számát (de 0 alá ne csökkentse).
     - Írj egy atkoltozik absztrakt metódust is, ami egy Szoba és egy egész szám paraméterrel rendelkezik, és visszatérése void.
   - Kézsítsd el az alábbi 3 osztályt:
     - Egyagyas, ami a Szoba osztályból származik. Konstruktora ne kérjen paramétert, és egy 1 fős szobát hozzon létre az ősosztály konstruktora segítségével. Az atkoltozik metódus Ketagyas illetve Lakosztaly típusú szobáka (objektum típusának eldöntése: instanceof segítségével) költözhessen át a megadott számú ember, ha a célszobában

van elég szabad hely. A két objektum vonatkozó adattagjait módosítsd ennek megfelelően. A szobának legyen toString metódusa, ami az ősosztály által kiírtak mellett azt is megmondja, hogy ez egy egyágyas szoba.
  - Ketagyas, ami a Szoba osztályból származik, és implementalja a Kedvezmenyes. Konstruktora ne kérjen paramétert, és egy 2 fős szobát hozzon létre az ősosztály konstruktora segítségével. Kétágyas szobára kedvezmény kérhető, ha csak 1 ember lakik benne, ilyenkor a bérleti díjat változtassa az egyágyasnak díjnak megfelelően. Az atkoltozik metódus Lakosztaly típusú szobáka (objektum típusának eldöntése: instanceof segítségével) költözhessen át a megadott számú ember, ha a célszobában van elég szabad hely. A két objektum vonatkozó adattagjait módosítsd ennek megfelelően. A szobának legyen toString metódusa, ami az ősosztály által kiírtak mellett azt is megmondja, hogy ez egy kétágyas szoba.
  - Lakosztaly, ami a Szoba osztályból származik. Konstruktora a lakók számát kérje paraméterként, és a megadott férőhellyel hozza létre a szobát az ősosztály konstruktora segítségével. Lakosztályból nem költözik ki senki, az atkoltozik függvény ezt közölje a felhasználóval. A szobának legyen toString metódusa, ami az ősosztály által kiírtak mellett azt is megmondja, hogy ez egy lakosztály.
- Kézsíts egy Panzio futtatható osztályt, amiben:
  - legyen egy statikus szobatKiad függvény. A függvény egy fájl elérési útját várja paraméterül. A metódus feladata, hogy a fájlból beolvasott sorokat feldolgozza, és létrehozzon belőlük Egyagyas, Ketagyas vagy Lakosztaly objektumpéldányokat, amiket egy közös tárolóba ment le.
  - legyen egy statikus berel metódus is, ami végigmegy a tárolón, és kiírja konzolra a rajta lévő szobákat. Ha egy szobára kérhető kedvezmény, úgy előtte tegye ezt meg. Minden esetleges kivételt (főleg FileNotFoundException és IOException) kezelj le vagy kivétel specifikációval, vagy try blokkban!
  - Ha hozott létre Egyagyas szobát és Lakosztalyt is ilyen módon a függvény, úgy csökkentsd az első létrehozott Lakosztaly lakóinak számát 1-el, és költöztesd át az első létrehozott Egyagyas szobából a lakót ide.
  - A main metódusban hívd meg a szobatKiad függvényt egy parancssori argumentumból bekért elérési úttal, majd hívd meg a berel metódust is.

  Egy minta fájl felépítése az alábbi: egyagyas ketagyas lakosztaly;10

```java
package panzio;

public interface Arak {
    public static final int EGYAGYAS = 8000;
    public static final int KETAGYAS = 12000;
}
```

Arak.java

```java
package panzio;
```

```
2
3  public interface Kedvezmenyes {
4
5      void kedvezmenytKer();
6  }
```

Kedvezmenyes.java

```
1  package panzio;
2
3  public abstract class Szoba implements Arak {
4
5      protected int berletiDij, fekvohely, lakok;
6
7      public Szoba(int lakok) {
8          this.lakok = lakok;
9          this.fekvohely = lakok;
10         if (lakok == 1) {
11             this.berletiDij = EGYAGYAS;
12         } else {
13             this.berletiDij = KETAGYAS + (lakok - 2) * EGYAGYAS;
14         }
15     }
16
17     @Override
18     public String toString() {
19         return "Szoba{" + "berletiDij=" + berletiDij + ",
               fekvohely=" + fekvohely + ", lakok=" + lakok + '}';
20     }
21
22     public void kikoltozik(int hany) {
23         lakok -= hany;
24         if (lakok < 0) {
25             lakok = 0;
26         }
27     }
28
29     public abstract void atkoltozik(Szoba sz, int hany);
30
31 }
```

Szoba.java

```java
package panzio;

public class Egyagyas extends Szoba {

    public Egyagyas() {
        super(1);
    }

    @Override
    public void atkoltozik(Szoba sz, int hany) {
        if (sz instanceof Ketagyas || sz instanceof Lakosztaly) {
            int szabad = sz.fekvohely - sz.lakok;
            if (szabad >= hany) {
                sz.lakok += hany;
                kikoltozik(hany);
            }
        }
    }

    @Override
    public String toString() {
        return "Egyagyas" + super.toString();
    }

}
```

Egyagyas.java

```java
package panzio;

public class Ketagyas extends Szoba implements Kedvezmenyes {

    public Ketagyas() {
        super(2);
    }

    @Override
    public void atkoltozik(Szoba sz, int hany) {
        int szabad = sz.fekvohely - sz.lakok;
        if (sz instanceof Lakosztaly && szabad >= hany) {
            sz.lakok += hany;
```

```
            kikoltozik(hany);
        }
    }

    @Override
    public void kedvezmenytKer() {
        if (lakok == 1) {
            berletiDij = EGYAGYAS;
        }
    }

    @Override
    public String toString() {
        return "Ketagyas" + super.toString();
    }

}
```

Ketagyas.java

```
package panzio;

public class Lakosztaly extends Szoba{

    public Lakosztaly(int lakok) {
        super(lakok);
    }

    @Override
    public void atkoltozik(Szoba sz, int hany) {
        System.out.println("Bocsi, nem megyunk mi sehova!");
    }

    @Override
    public String toString() {
        return "Lakosztaly " + super.toString();
    }


}
```

Lakosztaly.java

```
1  package panzio;
2
3  import java.util.*;
4  import java.io.*;
5
6  public class Panzio {
7
8      public static List<Szoba> szobak = new ArrayList<Szoba>();
9
10     public static void main(String[] args) {
11         try {
12             szobatKiad("input.txt");
13             berel();
14         } catch (Exception ex) {
15             System.out.println("ajajj: " + ex);
16         }
17     }
18
19     public static void berel() {
20         for (Szoba szoba : szobak) {
21             if (szoba instanceof Ketagyas) {
22                 ((Ketagyas) szoba).kedvezmenytKer();
23             }
24             System.out.println(szoba);
25         }
26     }
27
28     public static void szobatKiad(String path) throws IOException
           {
29         BufferedReader br = new BufferedReader(new FileReader(path
               ));
30         String line = br.readLine();
31         while (line != null) {
32             String[] conts = line.split(";");
33             switch (conts[0]) {
34                 case "egyagyas":
35                     szobak.add(new Egyagyas());
36                     break;
37                 case "ketagyas":
38                     szobak.add(new Ketagyas());
```

```
39                      break;
40                  case "lakosztaly":
41                      szobak.add(
42                              new Lakosztaly(Integer.parseInt(conts
                                    [1])));
43                      break;
44              }
45              line = br.readLine();
46          }
47          br.close();
48      }
49
50  }
```

Panzio.java