Java tutorial

Be- és kimenet kezelése

A legelső program

- Üdvözlő szöveg kiírása a képernyőre

```
class Hello {
  public static void main( String args[] ){
    System.out.println("Hello World!");
  }
}
```

Hogyan használjunk fájlokat?

- Triviális megoldás: a szabványos bemenet/kimenet átirányításával
- Az operációs rendszer szolgáltatása
- Az üdvözlő szöveget ki akarjuk írni a hello.txt nevű fájlba:

```
java Hello >hello.txt
```

Ha egy programban több fájl van

- Akkor a programon belülről is kellene tudni használni őket
- Van egy speciális könyvtár erre a célra:

  a java.io csomag
- Nem csak fájlok kezelése, hanem általában bemenet és kimenet

Bemenet és kimenet absztrakciója

- Bemenet: konzol, fájl, hálózati kapcsolat, adatbázis egy rekordja, stb.

  adatokat lehet egymás után olvasni róla
- Kimenet: képernyő, fájl, nyomtató, hálózati kapcsolat, adatbázis egy rekordja, stb.

  adatokat lehet egymás után kiírni rá
- „Szekvenciális input/output fájl”
- Csatorna (Stream)
- Csatornaobjektumok

Mire kell ez?

- Ugyanazokat a műveleteket lehet használni
- Egy jól megszervezett könyvtár: java.io

- Célok:
  - egyszerűség
  - rugalmasság
  - kifejezőerő

Java tutorial

Ismét az a legelső program

- Üdvözlő szöveg kiírása a képernyőre

```
class Hello {
  public static void main( String args[] ){
    System.out.println("Hello World!");
  }
}
```

Ugyanaz fájlba:

- Üdvözlő szöveg kiírása egy fájlba

```
class Hello {
  public static void main( String args[] ){
    System.out.println("Hello World!");
  }
}
```

Ugyanaz fájlba:

- Üdvözlő szöveg kiírása egy fájlba

```
class Hello {
  public static void main( String args[] ){
    FileWriter fw = new FileWriter("a.txt");
    System.out.println("Hello World!");
  }
}
```

Ugyanaz fájlba:

- Üdvözlő szöveg kiírása egy fájlba

```
class Hello {
  public static void main( String args[] ){
    FileWriter fw = new FileWriter("a.txt");
    PrintWriter pw = new PrintWriter(fw);
    System.out.println("Hello World!");
  }
}
```

Ugyanaz fájlba:

- Üdvözlő szöveg kiírása egy fájlba

```
class Hello {
  public static void main( String args[] ){
    FileWriter fw = new FileWriter("a.txt");
    PrintWriter pw = new PrintWriter(fw);
    System.out.println("Hello World!");
  }
}
```

Ugyanaz fájlba:

- Üdvözlő szöveg kiírása egy fájlba

```
class Hello {
  public static void main( String args[] ){
    FileWriter fw = new FileWriter("a.txt");
    PrintWriter pw = new PrintWriter(fw);
    pw.println("Hello World!");
  }
}
```

Ugyanaz fájlba:

- Üdvözlő szöveg kiírása egy fájlba

```
class Hello {
  public static void main( String args[] ){
    FileWriter fw = new FileWriter("a.txt");
    PrintWriter pw = new PrintWriter(fw);
    pw.println("Hello World!");
    pw.close();
  }
}
```

Ugyanaz fájlba:

- Üdvözlő szöveg kiírása egy fájlba

```
import java.io.*;
class Hello {
  public static void main( String args[] ){
    FileWriter fw = new FileWriter("a.txt");
    PrintWriter pw = new PrintWriter(fw);
    pw.println("Hello World!");
    pw.close();
  }
}
```

Ugyanaz fájlba:

- Üdvözlő szöveg kiírása egy fájlba

```
import java.io.*;
class Hello {
  public static void main( String args[] )
  throws IOException {
    FileWriter fw = new FileWriter("a.txt");
    PrintWriter pw = new PrintWriter(fw);
    pw.println("Hello World!");
    pw.close();
  }
}
```

Java tutorial

Ugyanaz fájlba:

- Üdvözlő szöveg kiírása egy fájlba

```
import java.io.*;
class Hello {
  public static void main( String args[] )
  throws IOException {
    FileWriter fw = new FileWriter("a.txt");
    PrintWriter pw = new PrintWriter(fw);
    pw.println("Hello World!");
    pw.close();
  }
}
```

Ugyanaz fájlba:

- Üdvözlő szöveg kiírása egy fájlba

```
import java.io.*;
class Hello {
  public static void main( String args[] )
  throws IOException {
    FileWriter fw = new FileWriter("a.txt");
    PrintWriter pw = new PrintWriter(fw);
    pw.println("Hello World!");
    pw.close();
  }
}
```

Java tutorial

System.out

- Ez egy predefinit csatorna
- Szabványos kimenet
- PrintStream
  - van neki println() művelete
  - olyasmi, mint a PrintWriter
- Van még System.in és System.err is

System.out.println("Hello");

A java.io csomag tartalma

- Sok pici osztály
- Mindegyik egy kis funkcionalitást valósít meg
- Könnyen komponálhatók

- Van néhány nagyon fontos osztály, a többi már haladóknak való... :-)

Az osztályok rendszerezése

- Különböző szempontok szerint csoportosíthatjuk az osztályokat
- Szimmetria
- Az osztályok neve sokat elárul
  - A nevek névkomponensekből épülnek fel, amelyek segítik az osztály hovatartozásának megállapítását
- Az osztályok hierarchiája is követi a logikai tagolást

Az osztályok csoportosítása

- Három szempont szerint lehet
  1. irány
     - bemenet
     - kimenet
  2. funkció
     - tárolás módjának specifikálása
     - extra funkcionalitás hozzáadása
  3. szervezés
     - bájt
     - karakter

Java tutorial

1. A csatornák iránya

- Lehet bemeneti és kimeneti
- Bemeneti: amiről olvasni lehet
  InputStream, Reader
- Kimeneti: amire írni lehet
  OutputStream, Writer
- Szimmetria: például
  FileInputStream, FileReader,
  FileOutputStream, FileWriter

2. A csatornaosztályok funkciója

2a. A tárolás módjának specifikálása
- hol vannak az adatok
- pl. FileWriter (fájlban)

2b. Extra funkcionalitás hozzáadása
- hogyan szeretnénk piszkálni az adatokat
- pl. PrintWriter (println() metódussal)

2a. A tárolás módjának specifikálása

- Honnan olvassuk az adatokat, vagy hova írjuk azokat
- FileReader, CharArrayReader, StringReader, PipedReader
- C-ben: scanf, fscanf, sscanf

- Szabványos műveletek minimális funkcionalitást biztosítanak

2b. Extra funkcionalitás hozzáadása

- Bonyolultabb műveletek külön osztályban
- Önmagukban egyszerűek
- Komponálhatók

- **Szűrők**

- Például a PrintWriter, BufferedReader, stb.

3. Az adatok szervezése

- Bájtszervezésű
  - A különböző karakterkódolási szabványok figyelmen kívül hagyása
  - Input- vagy OutputStream
- Karakterszervezésű
  - A különböző karakterkódolási szabványokból származó eltérések kezelése
  - Reader vagy Writer

3. Az adatok szervezése

- A karakterkódolási szabványok:
  - ASCII (7 bit)
  - extended ASCII, EBCDIC, ISO Latin-1, ISO Latin-2, Windows Latin-*, Mac, IBM, stb. (1 bájt)
  - Java: Unicode (2 bájt)
- Hordozhatóság, i18n, elosztott alkalmazások: a szabványok explicit kezelése
- Java 1.1-től
- Sok régi osztály/művelet elavult (deprecated)

Karakterszervezésű csatornák

- Javában a karakterek (char, String) két bájtosak
- Az operációs rendszerek és a nem Java alkalmazások többségében csak egy bájtosak (pl. szövegfájlok)
- Hogyan lehet a leképezést megadni a kettő között?
  - Karakterkódolási szabványok

Csatornaosztályok

- A csatornaosztályok hierarchiája és elnevezése követi a rendszert
- Könnyen kitalálható, hogy melyik osztály mire való
- Négy bázisosztály, és a többi ezeknek a leszármazottja

Java tutorial

A négy bázisosztály

- InputStream
  - bemeneti bájtcsatorna
- OutputStream
  - kimeneti bájtcsatorna
- Reader
  - bemeneti karaktercsatorna
- Writer
  - kimeneti karaktercsatorna

Leszármazottak

- A név tükrözi, hogy melyikből
- Szimmetria

- FileInputStream, FileOutputStream, FileReader, FileWriter
- BufferedInputStream, BufferedOutputStream, BufferedReader, BufferedWriter
- ByteArrayInputStream, ByteArrayOutputStream, CharArrayReader, CharArrayWriter

Java tutorial

Alapfunkcionalitás

- A négy bázisosztályban van specifikálva
- Ez a négy osztály absztrakt
- A leszármazottak valósítják meg a funkcionalitást
- Pl. a FileInputStream egy olyan InputStream, ami az alap csatorna-funkcionalitást fájlokon valósítja meg

Bemeneti csatornák

- Csatorna megnyitása
- Csatorna lezárása
- Olvasás a csatornáról

- Csatornán található adatok mennyiségének lekérdezése
- Könyvjelző mechanizmus

Kimeneti csatornák

- Csatorna megnyitása
- Csatorna lezárása
- Írás a csatornára
- Buffer ürítése (flush-olás)

Csatornák megnyitása

- A megfelelő osztályú csatorna objektum létrehozásával
- Ezen az objektumon végezhetők el a további csatornaműveletek

```
FileInputStream fin =
          new FileInputStream("a.txt");

InputStream fin =
          new FileInputStream("a.txt");
```

Csatornák bezárása

- A close() művelet segítségével
- Ne felejtsük el! Főleg kimenet csatornáknál fontos: magában foglalja a bufferelt adatok tényleges kiírását (flush) is.

```
fin.close();
```

Példa

```
import java.io.*;
class IOPróba {
  public static void main( String args[] )
  throws IOException {
    InputStream fin =
          new FileInputStream("a.txt");
    OutputStream fout =
          new FileOutputStream("b.txt");
    /* itt most csinálhatnék valamit... */
    fin.close(); fout.close();
  }
}
```

Java tutorial

Beolvasás

- read() műveletek segítségével
- Majdnem ugyanaz InputStream-ek és Reader-ek esetén
- háromfajta read() művelet

read()-ek: InputStream

- Egy adat beolvasása
  `int read() throws IOException`
  – egy bájtot beolvas; `-1`, ha vége
- Egy tömbnyi adat beolvasása
  `int read( byte[] b ) throws IOException`
  – olvas b-be; visszaadja a beolvasott bájtok számát
- Egy résztömbnyi adat beolvasása
  `int read( byte[] b, int off, int len ) throws IOException`
  – off-tól len hosszan olvas

read()-ek: Reader

- `int read() throws IOException`
- `int read( char[] c ) throws IOException`
- `int read( char[] c, int off, int len ) throws IOException`

- `int read() throws IOException`
- `int read( byte[] b ) throws IOException`
- `int read( byte[] b, int off, int len ) throws IOException`

- Ugyanaz, csak **`byte[]`** helyett **`char[]`**

Java tutorial

Kiírás

- write() műveletek segítségével
- Majdnem ugyanaz OutputStream-ek és Writer-ek esetén
- Szimmetrikus a beolvasó read() műveletekkel
- háromfajta alap write() művelet
  - plussz még egy kis kiegészítés

write()-ok: OutputStream

- Egy adat kiírása

```
void write( int i ) throws IOException
```

  - az i-nek a legalsó bájtját írja ki

```
int read() throws IOException
```

- Egy tömbnyi adat kiírása

```
void write( byte[] b ) throws IOException
int   read( byte[] b ) throws IOException
```

- Egy résztömbnyi adat kiírása

```
void write( byte[] b, int off, int len )
  throws IOException
int   read( byte[] b, int off, int len )
  throws IOException
```

write()-ok: Writer

- `void write( int i ) throws IOException`
- `void write( char[] c ) throws IOException`
- `void write( char[] c, int off, int len ) throws IOException`

write()-ok: Writer

- `void write( int i ) throws IOException`
  - az i legalsó két bájtját írja ki
- `void write( char[] c ) throws IOException`
- `void write( char[] c, int off, int len ) throws IOException`

write()-ok: Writer

- `void write( int i ) throws IOException`
- `void write( char[] c ) throws IOException`
- `void write( char[] c, int off, int len ) throws IOException`
- `void write( String s ) throws IOException`
- `void write( String s, int off, int len ) throws IOException`

Példa: másolás

```
static void másol
( InputStream in, OutputStream out )
throws IOException {
    int b;
    while( (b=in.read()) != -1 )
      out.write(b);
    out.flush();
}
```

Összerakva:

```
import java.io.*;
class cp {
  public static void main( String args[] )
  throws IOException {
    InputStream fin =
          new FileInputStream(args[0]);
    OutputStream fout =
          new FileOutputStream(args[1]);
    másol(fin,fout);
    fin.close(); fout.close();
  }
  static void másol(...) ... {...}
}
```

Feladat

- Adott két fájl, "a.txt" és "b.txt". Fésüljük össze a két fájl tartalmát a "c.txt" fájlba. Először egy bájt az "a.txt"-ből, aztán egy a "b.txt"-ből, aztán megint egy az "a.txt"-ből, stb. Ha valamelyik fájl végetér, a másik fájl maradékát másoljuk a "c.txt" végére.

Visszatérve még egy percre...

```
static void másol
( InputStream in, OutputStream out )
throws IOException {
    int b;
    while( (b=in.read()) != -1 )
      out.write(b);
    out.flush();
}
```

- Buffer ürítése: az adatok ténylegesen kiíródjanak a fizikai adathordozóra

Java tutorial

Másolás tömbbel

```
static int BlokkMéret = 100;

public static void másolBlokkonként
( InputStream in, OutputStream out )
throws IOException {
  byte[] b = new byte[BlokkMéret];
  int hossz;
  while( (hossz=in.read(b)) == BlokkMéret )
    out.write(b);
  if( hossz != -1 ) out.write(b,0,hossz);
  out.flush();
}
```

- egyszerre 100 bájtot másolunk át

Feladat

- Az előző feladatra adott megoldást módosítsd úgy, hogy az a.txt és a b.txt fájlokból 10 bájtos blokkokat másoljon a c.txt fájlba!

Java tutorial

Véget ért és üres csatornák

- Két teljesen különböző dolog!
- Véget ért: ha üres, és üres is marad.
- Üres: lehet, hogy még nem ért véget, csak éppen nem érkezett még meg az adat
- Gondoljunk egy hálózati kapcsolatra
  - Pl. a szerver várja, hogy a kliens küldjön adatot

Mi a különbség a beolvasó műveletek számára?

- Ha véget ért egy csatorna, a beolvasás befejeződik, és a visszatérési érték jelzi, hogy nincs több adat.
  - első read: -1 a visszatérési érték
  - másik két read: a beolvasott adatok számát adja vissza
- Ha üres a csatorna, a beolvasó művelet **blokkolódik**, várja, hogy érkezzen adat.
  - a végrehajtási szál felfüggesztődik, nem fut tovább

Példák

- Véget ért csatorna: az előző példákban a ciklusból kiléptünk, amikor végetért a bemenet
- Blokkolt beolvasás: a program vár, amíg be nem írunk egy sort. Ekkor a shell odateszi az adatokat a program szabványos bemenetére.

```
public static void main( String args[] )
throws IOException {
  int i = System.in.read();
  System.out.write(i); System.out.flush();
}
```

Különbség a write és a print között

```
public static void main( String args[] )
throws IOException {
  int i = System.in.read();
  System.out.write(i);
  System.out.println(" " + i);
}
```

- Kiírjuk a beolvasott betűt (bájtot) úgy, ahogy beolvastuk, majd utána a kódját.
- print: kinyomtatás szöveges formában

Mi van még a fájlokon kívül?

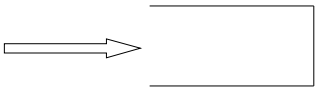
- Csatornák segítségével nem csak fájlokból lehet olvasni, és fájlokba írni.
- Memória, hálózati kapcsolat, nyomtató, stb.
- Mindezekről kicsit később ejtünk majd szót...

Csatornák feladat szerinti csoportosítása

- Alapfunkcionalitás megvalósítása különböző adathordozókon (pl. fájlokon)
- **Extra funkcionalitással való ellátás: szűrők**

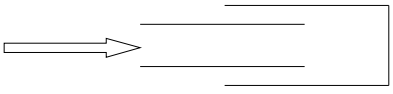
Java tutorial

Szűrők



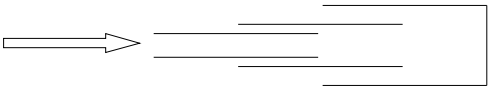
Kimeneti csatorna, pl. egy fájl

Szűrők



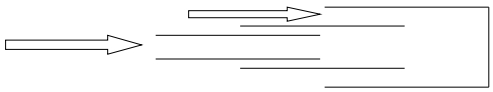
Egy szűrőn keresztül írunk,
ami extra funkcionalitást biztosít
(pl. bufferelés)

Szűrők



A szűrő is egy csatorna, amit egy újabb szűrővel láthatunk el. Komponálhatjuk a szűrőket, hogy bonyolultabb viselkedést állíthassunk elő. (Pl. bufferelés és print-elés egyszerre)

Szűrők

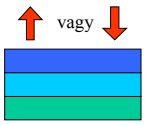


A szűrő(ke)t megkerülve is írhatunk a külső csatornára, de ez eléggé veszélyes. (Gondoljunk pl. arra, hogy a szűrő a bufferelés...)

Szóhasználat

- A szűrőket mindig egy már meglévő **csatorna fölé** hozzuk létre.
- A szűrő csatorna konstruktorának paraméterként meg kell adni a szűrt csatornát

```
FileWriter fw = new FileWriter("a.txt");
PrintWriter pw = new PrintWriter(fw);
```



A legfontosabb szűrők

- Bufferelés
- Adattípus-értékek beolvasása és kiírása
- Szöveges formában történő kiírás („nyomtatás”)

Java tutorial

Az első példánk volt: szöveges kiírás fájlba

```
import java.io.*;
class Hello {
  public static void main( String args[] )
  throws IOException {
    FileWriter fw = new FileWriter("a.txt");
    PrintWriter pw = new PrintWriter(fw);
    pw.println("Hello World!");
    pw.close();
  }
}
```

A FileWriter objektumnak nincs println művelete!

Az első példánk volt: szöveges kiírás fájlba

```
import java.io.*;
class Hello {
  public static void main( String args[] )
  throws IOException {
    FileWriter fw = new FileWriter("a.txt");
    PrintWriter pw = new PrintWriter(fw);
    pw.println("Hello World!");
    pw.close();
  }
}
```

A szűrő konstruktorának átadjuk fw-t!

Az első példánk volt: szöveges kiírás fájlba

```
import java.io.*;
class Hello {
  public static void main( String args[] )
  throws IOException {
    FileWriter fw = new FileWriter("a.txt");
    PrintWriter pw = new PrintWriter(fw);
    pw.println(42*Integer.parseInt(args[0]));
    pw.close();
  }
}
```

Egy PrintWriter szűrővel kinyomtathatunk számokat!

Az első példánk volt: szöveges kiírás fájlba

```
import java.io.*;
class Hello {
  public static void main( String args[] )
  throws IOException {
    FileWriter fw = new FileWriter("a.txt");
    PrintWriter pw = new PrintWriter(fw);
    pw.println(args.length > 0);
    pw.close();
  }
}
```

Vagy logikai értékeket, stb.

Szöveges kiírás

- PrintStream és PrintWriter
- Nevezetesek a System.out és System.err
- Persze nem csak fájlok fölé hozhatjuk létre...
- Kinyomtathatunk adattípus-értékeket is, és objektumokat is – a toString() metóduson keresztül
- print: sima kinyomtatás
  println: mint print, plussz soremelés

```
System.out.print(42);
System.out.print("42");
```

Java tutorial

Bináris kiírás

- DataOutputStream
- Adattípus-értékek bináris kiírása
  - Ha nem egyszerűen bájtokat szeretnénk kiírni...
  - Pl. kiírhatjuk egyszerre mind a négy bájtot, ami egy float típusú értékhez kell

```
FileOutputStream fout = new
  FileOutputStream("a.dat");
DataOutputStream dout = new
  DataOutputStream(fout);
dout.writeFloat((float)12.0);
```

Bináris beolvasás

- DataInputStream
- Ugyanaz visszafelé is megy: beolvasunk négy bájtot, és rekonstruáljuk belőle a float típusú értéket

```
DataInputStream din =
  new DataInputStream(
    new FileInputStream("a.dat") );
float f = din.readFloat();
```

Lehetőségek

- writeBoolean
- writeByte
- writeShort
- writeChar
- writeInt
- writeLong
- writeFloat
- writeDouble
- writeUTF
- readBoolean
- readByte
- readShort
- readChar
- readInt
- readLong
- readFloat
- readDouble
- readUTF

Mire jó ez?

- Adatok tömör tárolását teszi lehetővé a bináris formátum
- A DataOutputStream és a DataInputStream segítségével egyszerűen lehet adattípus-értékeket elmenteni és visszaolvasni
- Platform-független megoldás: a bináris formátumot a Java pontosan specifikálja (big-endian vs. little-endian probléma)

Java tutorial

Feladat

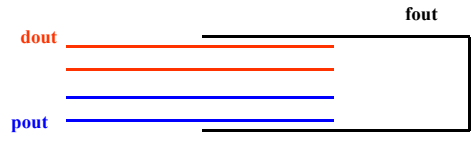
- A mátrix osztályt egészítsük ki bináris elmentő és beolvasó műveletekkel.
- Írjunk hozzá toString() metódust is, hogy megkönnyítsük a képernyőre való kiírást.
- Írjunk egy főprogramot, ami elment egy egységmátrixot.
- Írjunk egy másik főprogramot, ami egy fájlból beolvasott mátrixhoz hozzáadja a csupa egyes mátrixot, és visszaírja a fájlba.

Kérdés

- Melyik hány bájtot ír ki?
  - `dout.writeInt(12)`
  - `pout.print(12)`

Különbség a szöveges és a bináris formátum között

```
FileOutputStream fout =
  new FileOutputStream("okos.txt");
DataOutputStream dout =
  new DataOutputStream(fout);
PrintStream      pout =
  new PrintStream(fout);
```



Különbség a szöveges és a bináris formátum között

```
FileOutputStream fout =
  new FileOutputStream("okos.txt");
DataOutputStream dout =
  new DataOutputStream(fout);
PrintStream      pout =
  new PrintStream(fout);
dout.writeInt( 1869311859);   dout.flush();
pout.print(    1869311859);   pout.flush();
fout.write(    1869311859);   fout.flush();
```

- A számok többségénél tényleg tömörebb a bináris forma...

PrintStream inverze?

- DataInputStream - DataOutputStream
  beolvasás - kiírás
- ??? - PrintStream/PrintWriter

- Nincs olyan csatornaosztály, aminek a segítségével egyszerűen lehetne szöveges formátumban elmentett adattípus-értékeket visszaolvasni
- Megoldás: BufferedReader - mindjárt...
- Másik: StreamTokenizer (haladó...)

Bufferelés

- BufferedInputStream, BufferedReader
  BufferedOutputStream, BufferedWriter
- Ezek a csatornák bufferelnek
- Jó esetben az oprendszer is bufferel...

```
InputStream bin =
  new BufferedInputStream(
     new FileInputStream("a.txt")  );
```

Feladat

- Játsszuk ki a bufferelést! Írjunk olyan programot, ami összevissza ír egy fájlba.

Szűrők komponálása

- A bufferelést gyakran használjuk egyéb szűrővel együtt, pl. Data*Stream

```
FileInputStream     fin =
     new FileInputStream(távoli_fájl);
BufferedInputStream bin =
     new BufferedInputStream(fin);
DataInputStream     din =
     new DataInputStream(bin);
float összeg = 0.0F;
for( int i=0; i<100; i++ )
     összeg += din.readFloat(); }
```

Java tutorial

Még egy fontos dolog...

- BufferedReader
- Sorok olvasása szöveges állományból
- A metódus:
  `String readLine() throws IOException`
- Soremelésig olvas, a soremelés jelet nem adja vissza
  - Soremelés jel: `\r` `\n` `\r\n`
- Csatorna vége esetén: `null`

Példa
Interaktív alfanumerikus program

```
public static void main( String args[] )
throws IOException {
  BufferedReader r =
     new BufferedReader(
          new InputStreamReader(System.in));
  System.out.println("Üssön be egy számot: ");
  String s = r.readLine();
  int n = Integer.parseInt(s);
  System.out.println("A szám négyzete: " + n*n);
}
```

Feladat

- Írjunk olyan metódust a mátrix osztályhoz, amelynek segítségével szöveges formátumból be lehet olvasni egy mátrixot. A mátrix elemei mind külön sorban legyenek!
- Használj BufferedReader osztályt egy FileReader fölé húzva!

Java tutorial

Közvetlen elérésű fájlok

- Sokan szeretnek pozícionálni egy fájlban...
- Ez nem megy a csatornaosztályok esetén
- RandomAccessFile
- Meg lehet nyitni egy fájlt, és lehet olvasni ("r") vagy írni/olvasni ("rw")
- Műveletek: mint DataInputStream és DataOutputStream, de együtt

Példa

- Minden háromhatványadik számot inkrementálunk.

```
RandomAccessFile f =
     new RandomAccessFile("adatok.dat","rw");
int háromhatvány = 1;
while( háromhatvány * 4 < f.length() ){
     f.seek(háromhatvány * 4);
     int adat = f.readInt();
     f.seek(háromhatvány * 4);
     f.writeInt(adat+1);
     háromhatvány *= 3;
}
f.close();
```

Feladat

- Indexelt fájlkezelés
- Az indexelt fájl 1024 hosszúságú, double értékeket tartalmazó sorozatokat tartalmaz.
- Az indexeléshez használjunk lexikografikus rendezést.

Java tutorial

Haladó kurzushoz

Mik vannak még a fájlokon kívül?

- A feladat szerinti csoportosítás szerint: honnan olvasunk, illetve hova írunk
- Lehetőségek a java.io csomagban
  - fájlok
  - csövek
  - bájt- és karaktertömbök, sztringek
  - bájtcsatorna felett definiált karaktercsatornák
  - összefűzött csatornák

Mik vannak még a fájlokon kívül?

- A feladat szerinti csoportosítás szerint: honnan olvasunk, illetve hova írunk
- Lehetőségek a java.io csomagban
  - fájlok
  - csövek
  - bájt- és karaktertömbök, sztringek
  - bájtcsatorna felett definiált karaktercsatornák
  - összefűzött csatornák

Csövek

- Egy PipedInputStream-et és egy PipedOutputStream-et összekapcsolhatunk egymással.
- Amit a pout-ra írunk, megjelenik az pin-en.
- Mire jók? Konkurrens végrehajtási szálak egymás közötti kommunikációjára, pl.
- Később lesz róluk bővebben szó.

Memória műveletek

- Az adatok, amiket végigolvasunk, vagy amiket kiírunk, egy memóriacímtől kezdve, folytonosan helyezkednek el
- Például egy bájttömbben
  - ByteArrayInputStream, -OutputStream
- Vagy esetleg egy karaktertömbben
  - CharArrayReader, -Writer
- Vagy egy sztringben
  - StringReader, -Writer
  - StringBufferInputStream

Karaktertömbből olvasunk

```
char[] t = {'r', 'ó', 'k', 'a'};
CharArrayReader in = new
  CharArrayReader(t);
t[0] = 'f';
System.out.println(in.read()=='f');
```

"true"-t fog kiírni

Bájtcsatornák felett definiált karaktercsatornák

- Kapcsolat a kétféle szervezésű csatornaosztályok között
- A különböző karakterkódolási szabványok támogatása
- InputStreamReader, OutputStreamWriter
- Specifikálhatjuk, hogy melyik karakter melyik bájtra legyen leképezve

Karakterek leképezése bájtokra

- Egy byte típusú érték 256-féle lehet
- Karakterből jóval többféle van
- Karakterkódolási szabvány: mely karaktereket reprezentálja ez a 256 különböző érték
- Például a Unicode karakterek kétbájtosak
  - a char típus

Karakterkódolási szabványok

- ASCII - 7 bit
- Extended ASCII - 8 bit
- EBCDIC
- Latin-1, Latin-2
- Windows-os kódtáblák
- Mac-es kódtáblák

Java tutorial

Példa

```
Writer w = new
  OutputStreamWriter(System.out,"8859_2");
w.write(336);
w.flush();
System.out.write(336);
```

- 8859_2 ISO Latin-2
- A 336 Unicode kódú karakter az "Ő"
- A Latin-2 tartalmazza ezt a karaktert, a 256 lehetséges érték közül a 213 van hozzárendelve
- Tehát a System.out-ra a 213 bájt fog kiíródni
- Platform-függő, hogy mit fogunk látni
  - Pl. Latin-1 képernyő esetén "Õ" karaktert

ISO Latin-1

- Az ISO Latin-1 kódolásban szereplő karakterek a Nyugat-Európában használatos betűket (is) tartalmazzák
- Ezen karakterekhez az ISO Latin-1 ugyanazokat a számokat rendeli, mint a Unicode. Azaz: a Unicode kódtábla első 256 karaktere ugyanaz, mint a Latin-1
- Azaz a hullámos O betű Unicode kódja is 213

ISO Latin-2

- Tartalmazza azokat a betűket is, amelyeket csak mifelénk használnak, nevezetesen az ő, ű, Ő és Ű betűket
- Az előbbi példában: a 336 Unicode kódú karaktert, az Ő-t, és a 213 értékkel kapcsolja össze
- Ez a karakter nincs az ISO Latin-1 kódtáblában, abban a hullámos változat van a 213 értékhez kapcsolva

Példa

```
Writer w = new
  OutputStreamWriter(System.out,"8859_1");
w.write("tekn\u0150c");
w.close();
```

- A \u0150 karakter a 336-os, azaz az Ő betű
- Ez nincs a Latin-1-ben (8859_1)
- A kimeneten a ? karakternek megfelelő bájt íródik ki helyette
- **tekn?c**

Java tutorial

Másik példa

```
Writer w = new
  OutputStreamWriter(os,"MacCentralEurope");
w.write('Ú'); w.flush();   // w.write(218);
Reader r = new
  InputStreamReader(is,"Cp1250");
int i = r.read();
System.out.println(i);     // 328 jelenik meg
```

- A 218 Unicode kód az Ú betű
- A MacCentralEurope a 242 kódot rendeli hozzá
- A Windows Latin-2 (azaz a Cp1250) a 328 Unicode kódú karaktert rendeli a 242 értékhez
- Ez a n̂ betű

Tanulság

- Ugyanazzal a karakterkódolással érdemes beolvasni, mint amivel kiírtuk az adatokat
- Ha a Java programunk más programmal / más platformmal tartja a kapcsolatot, akkor is jól megy
- Ez a szabály platformfüggetlenséget is biztosít
- Két Java program közötti kommunikációnál is fontos (p. hálózatos program)

A Java által támogatott karakterkódolási szabványok

- ISO Latin-1, ISO Latin-2, ...
- Windows Latin-1, ...
- Mac kódolások
- IBM kódtáblák
- Távol-keleti szabványok (Big5, JIS, stb.)

- Mindnek megfelel egy sztring, amit a konstruktornak kell átadni. Pl. "8859_1"

Karakterszervezésű fájlcsatornák

```
• Reader fr = new FileReader("olvass.el");
• Reader fr = new InputStreamReader(
          new FileInputStream("olvass.el") );

• Reader fr = new InputStreamReader(
          new FileInputStream("olvass.el"),
                          "8859_2");
```

Java tutorial

Csatornák egymás után fűzése

- SequenceInputStream
- A konstruktorának megadunk több InputStream-et
- Ezeket egymás után végigolvashatjuk: ha az egyik véget ér, automatikusan ugrik a következőre

Példák

```
InputStream in = new SequenceInputStream(
        (new FileInputStream("papsajt")),
        System.in
    );

Vector v = new Vector(2);
v.addElement(new FileInputStream("papsajt"));
v.addElement(System.in);
InputStream in =
     new SequenceInputStream(v.elements());
```

- Végigolvashatjuk a "papsajt" fájlt, és folytatjuk az olvasást a szabványos bemenetről

Java tutorial

Miből lehet még olvasni, mire lehet még írni?

- Más könyvtárakban is vannak olyan osztályok, metódusok, amelyek segítségével csatornákhoz juthatunk hozzá
- Például a java.net.Socket osztálytól tudunk olyan bemeneti és kimeneti csatornákat beszerezni, amivel hálózati kapcsolaton keresztül lehet irkálni, olvasgatni...

Kiegészítések az alapfunkcionalitáshoz

- A két bemeneti csatorna alaposztálynak, az InputStream-nek és a Reader-nek további "szolgáltatásai"
- Könyvjelző mechanizmus
- Adatelérhetőség-vizsgálat

Adatelérhetőség-vizsgálat

- InputStream: int available()
- Reader: boolean ready()
- Meg lehet tudni, hogy a beolvasás blokkolódni fog-e. Az **`available`** csak becslést ad!

```
public static int méret( String fnév )
throws IOException {
  return
    (new FileInputStream(fnév)).available();
}
```

Könyvjelző mechanizmus

```
void mark(int readlimit) throws IOException
boolean markSupported()
void reset() throws IOException
```

- Betehetek egy könyvjelzőt az aktuális pozícióra
- ... amennyiben a csatornaosztály támogatja ...
- Visszaugrok a könyvjelzőig

Könyvjelző mechanizmus

```
void mark(int readlimit) throws IOException
boolean markSupported()
void reset() throws IOException
```

- Betehetek egy könyvjelzőt az aktuális pozícióra
  - Ha visszaugrok a reset-tel, akkor innen kezdve megint elolvashatom a csatorna tartalmát
  - A mark és a reset között max. "readlimit" bájtot olvashatok át...

Könyvjelző mechanizmus

```
void mark(int readlimit) throws IOException
boolean markSupported()
void reset() throws IOException
```

- Jónéhány csatornaosztály támogatja a könyvjelző mechanizmust, de jó sok nem

Példa

```
public static void kétszer
( ByteArrayInputStream in )
throws IOException {
     int c;
     in.mark(in.available());
     while( (c=in.read()) != -1 )
          System.out.println(c);
     in.reset();
     // még egyszer írjuk ki ugyanazt
     while( (c=in.read()) != -1 )
          System.out.println(c);
}
```

Java tutorial

Extra funkcionalitás: szűrők

- Már beszéltünk ezekről:
  - bufferelés
  - adattípus-értékek
  - szöveges „nyomtatás”
- Továbbiakat fogunk megismerni...

Még mielőtt rátérnénk...

- DataInputStream DataOutputStream
  String readUTF() void writeUTF(String s)
- PrintStream, PrintWriter kivételek

DataInputStream
String readUTF()

DataOutputStream
void writeUTF(String s)

- Unicode sztringek elmentése és visszaolvasása
- Platformfüggetlen
  - ún. UTF-8 formátumban
- Információvesztés nélkül
  - ugyanis a karakterszervezésű csatornák csak a nekik megfelelő karakterkódolási szabvány szerinti karaktereket tudják

Feladat

- Írjunk ki fájlba sztringeket és karaktertömböket az összes általunk ismert módon.
- Derítsük fel a különbségeket!

PrintStream, PrintWriter: kivételek

- A műveletek nem váltanak ki IOException kivételt gond esetén
- Helyette egy error-flag-et állítanak
- `boolean checkError()`

Extra funkcionalitás: további szűrők

- bemenet sorainak számolása
- adat visszatevése
- objektumok beolvasása / kiírása

- tömörítés
- stb.

A bemenet sorainak számolása

- LineNumberReader, LineNumberInputStream

```
LineNumberReader be =
          new LineNumberReader(
               new FileReader("borok.txt") );
String sor;
while( (sor = be.readLine()) != null ){
  if( sor.endsWith("bor") )
    System.out.println(be.getLineNumber()-1);
}
be.close();
```

Adat visszatevése a bemeneti csatornára

- PushbackReader, PushbackInputStream

```
PushbackReader be = new PushbackReader(
     new InputStreamReader(System.in), 2 );
int c = be.read();
while( (c == ' ')  || (c == '\t') ||
       (c == '\r') || (c == '\n') )
   c = be.read();
if( c != -1 ){
   be.unread(c);
   be.unread(' ');
}
```

Java tutorial

Objektumok tárolása

- ObjectInputStream, ObjectOutputStream
- kiterjesztik a Data*putStream osztályokat
- de nem csak adattípus-értékeket lehet elmenteni / visszaolvasni, hanem **objektumokat is**

```
void writeObject( Object obj )
                    throws IOException
Object readObject()
     throws OptionalDataException,
     ClassNotFoundException, IOException
```

Objektumok tárolása

- ObjectInputStream, ObjectOutputStream
- kiterjesztik a Data*putStream osztályokat
- de nem csak adattípus-értékeket lehet elmenteni / visszaolvasni, hanem **objektumokat is**

```
void writeObject( Object obj )
                    throws ...
Object readObject()
                    throws ...
```

Szerializáció

- Amikor egy objektumot elmentünk, azokat az objektumokat is el kell menteni, amelyekre hivatkozik...
- És ez így megy rekurzívan...
- Ha egy objektum a hivatkozottak között többször is előfordul, akkor is csak egyszer kell elmenteni!

Serializable

- Ezt a bonyolult feladatot a Java megcsinálja nekünk.
- Csak annyit kell tenni, hogy megvalósítjuk a Serializable interfészt
  - nem specifikál metódusokat...
  - csak oda kell írni az osztályunk definíciójába

Mi is történik?

- Metainformáció is elmentésre kerül
  - az objektum osztálya, verziószám, stb.
- Elmentődnek az objektum adattagjai
- Elmentődnek a hivatkozott objektumok is rekurzívan
- Ha az objektum már korábban el lett mentve, akkor helyette egy "mutató" lesz elmentve

Java tutorial

Példa

```
Vector v = new Vector();
v.add("Lennon");
v.add(new Integer(42));
ObjectOutputStream out =
     new ObjectOutputStream(
          new FileOutputStream("a.dat") );
out.writeObject(v);
out.close();

ObjectInputStream in =
     new ObjectInputStream(
          new FileInputStream("a.dat") );
v = (Vector) in.readObject();

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

Feladat

- Csináljunk egy főnököt két beosztottal. Mentsük el a főnököt.
- Egy másik programmal olvassuk be a főnököt (a két beosztottjával együtt).
- Az Alkalmazott és a Főnök osztályokban írjunk `toString()` metódust, hogy tudjuk tesztelni az előző két programot...
- Csak a hecc kedvéért... a főnök egyik beosztottja legyen saját maga.

Java tutorial

Finomhangolás

- Lehet szabályozni, hogy pontosan mi is mentődjön el az objektum által tárolt adatokból
- Több szinten nyílik lehetőség a szabályozásra
  - transient
  - read/writeObject felüldefiniálása
  - serialPersistantFields
  - Externalizable

Finomhangolás

- Lehet szabályozni, hogy pontosan mi is mentődjön el az objektum által tárolt adatokból
- Több szinten nyílik lehetőség a szabályozásra
  - **transient**
  - read/writeObject felüldefiniálása
  - serialPersistantFields
  - Externalizable

transient

- Egy módosítószó
  - mint pl. public, final, abstract, stb.
- Ha egy adattagot nem szeretnénk szerializáció során elmenteni, akkor elé írjuk, hogy transient
- Mire használjuk?
  - Egyszeri adat, pl. erőforrás-leíró (Frame)
  - Bizalmas adat
  - Redundáns információ

Java tutorial

A fájlrendszer fájljaihoz való hozzáférés

- A File osztályon keresztül történhet
- A File objektumok fájlneveket reprezentálnak
- Nem fájlokat: egy File objektumon keresztül nem lehet olvasni vagy írni egy fájlt. Arra ott vannak
  - a csatornák
  - a közvetlen elérésű fájlok

Mit lehet csinálni File objektumokkal?

- Lekérdezni, hogy létezik-e olyan nevű fájl
- Lekérdezni, hogy olvasható/írható-e
- Egy fájlt törölni
- Átnevezni
- Könyvtárat kilistázni
- stb.

- Ezek műveletek a fájlrendszeren...

Példa

```
File törlendő = new File("dobj.ki");
if( törlendő.canWrite() )
  törlendő.delete();
```

File objektum létrehozása

```
File könyvtár = new File("/usr/local/bin");
File ez_is_az = new File("/usr/local","bin");
File fájl = new File(könyvtár,"vim");

String elv = File.separator;
File könyvtár =
  new File(elv+"usr"+elv+"local"+elv+"bin");
```

Műveletekre példa

```
public boolean exists()
public boolean isFile()
public boolean renameTo( File cél )
public boolean mkdir()
public boolean isAbsolute()
public File getParentFile()
public long lastModified()
```

Feladat

- Listázzuk ki a tmp alkönyvtár összes txt kiterjesztésű fájljának nevét!
- Ehhez kicsit körül kell nézni az API dokumentációban...

Java tutorial

Szövegfeldolgozás

- StreamTokenizer
- Egy csatorna tartalmát tokenekre bontja

  `int nextToken() throws IOException`
  - számok
  - azonosítók
  - megjegyzések
- Egy csatorna fölé hozzuk létre, mint egy szűrőt (de ez nem csatorna)

Szavakra bontunk egy szöveget

```
import java.io.*;
class Szavakra {
  public static void main( String args[] )
  throws IOException {
    StreamTokenizer st =
       new StreamTokenizer(
          new InputStreamReader(System.in) );
    while( st.nextToken() !=
           StreamTokenizer.TT_EOF )
      if( st.ttype == StreamTokenizer.TT_WORD )
        System.out.println(st.sval);
  }
}
```

Próbáld ki!

Javítás

```
st.ordinaryChar(46);
st.slashStarComments(true);
```

- A pont írásjel közönséges karakter. A szövegben a C-stílusú megjegyzéseket figyelmen kívül akarjuk hagyni.

```
A kutya /* ez                A
egy állat */ ugat.           kutya
                             ugat
```

int-tek beolvasása szöveges formátumból

- Szöveg formátum, pl. szövegfájl
- int típusú értékeket tartalmaz
- Szeretnénk egy csatorna osztályt, amivel ilyet könnyű feldolgozni
- Szűrő csatornaként valósítjuk meg
  - A PrintStream „inverze”
- Segít a StreamTokenizer osztály

int-tek beolvasása szöveges formátumból

```
import java.io.*;
class NotExpectedTokenException extends Exception {
  public int token;
  public NotExpectedTokenException( int token )
            { this.token=token; }
}

public class TypedReader extends FilterReader {
  public TypedReader( Reader in ){ super(in); }

  public int scanInt()
  throws NotExpectedTokenException, IOException {...}

  public static void main( String[] args )
  throws IOException {...}
}
```

```
public int scanInt()
throws NotExpectedTokenException, IOException {

    StreamTokenizer st = new StreamTokenizer(in);
    st.ordinaryChar('/');      // nincs megjegyzésjel
    st.ordinaryChar('.');      // ne legyen tizedespont
    st.nextToken();
    if ( (st.ttype==StreamTokenizer.TT_NUMBER) &&
         (st.nval>=Integer.MIN_VALUE) &&
         (st.nval<=Integer.MAX_VALUE) )
            return (int)st.nval;
            // sikerült számként értelmezni
    else throw new NotExpectedTokenException(st.ttype);
  }
```

```
public static void main( String[] args )
throws IOException {

    TypedReader in = new TypedReader(
            new InputStreamReader(System.in) );

    int sum = 0;
    boolean endOfInts = false;
    while (!endOfInts) {
      System.out.print("Írjon be egy számot: ");
      try {   sum += in.scanInt();
              System.out.println();
      } catch (NotExpectedTokenException e)
              { endOfInts=true; }
    }
    System.out.println("\nAz összeg: " + sum);
  }
```

Feladat

- A mátrix osztályhoz írjunk olyan beolvasó műveletet, ami egy szövegfájlból olvas be egy mátrixot. A szövegfájlban nem feltétlenül soronként vannak a mátrix elemei, egy sorban lehet több is. Az elemeket fehér szóközök választják el. Ha ettől eltérő bemenetet tapasztalunk, váltsunk ki kivételt!

Java tutorial