Budapesti Műszaki és Gazdaságtudományi Egyetem

Java technológia

# Adat be- és kivitel

**Sipos Róbert**

**siposr@hit.bme.hu**

**2014. 03. 20.**

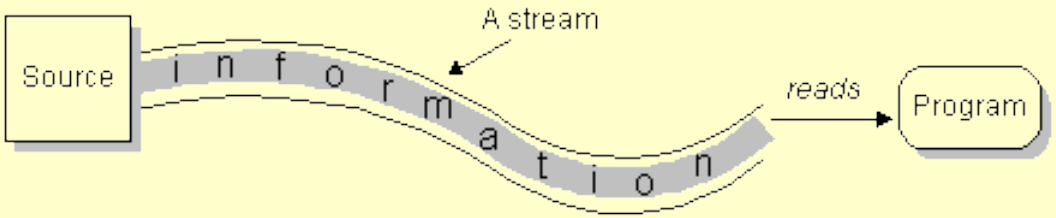
# Bevezetés

- **io (input/output)**: az a mechanizmus, ahogy egy program külső adatokat használ, illetve adatait külső erőforrásba továbbítja

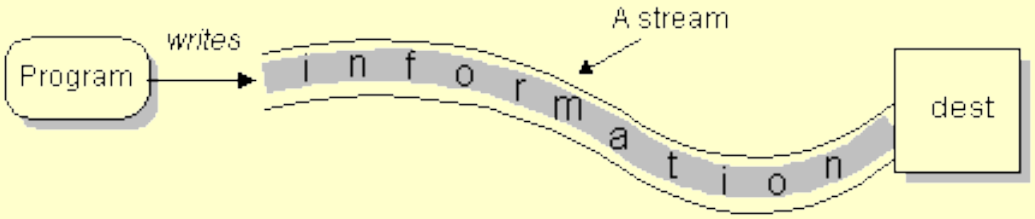

- Az adat **helye** lehet:
  - Fájl
  - Memória
  - Más program
  - Hálózat
- Az adat **típusa** lehet:
  - Karakteres adat
  - Bináris adat
  - Objektum, …

# IO menete I.

- Alapfogalom: stream, adatfolyam – szekvenciális hozzáférésű
- Bemeneti adatok = adatfolyam, stream




- Kimeneti adatok = adatfolyam, stream



# IO menete II.

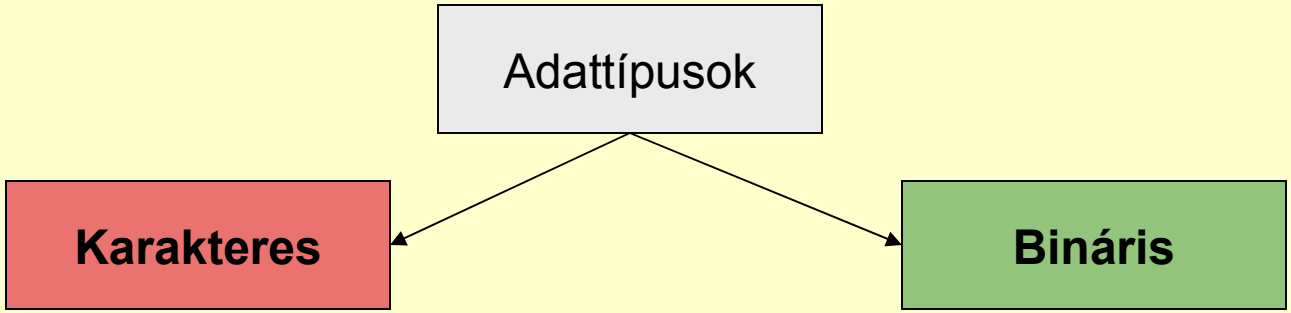
- Adatbevitel:

```
open a stream
while (more information)
  read information
close the stream
```

- Kivitel:

```
open a stream
while (more information)
  write information
close the stream
```

# IO típusok



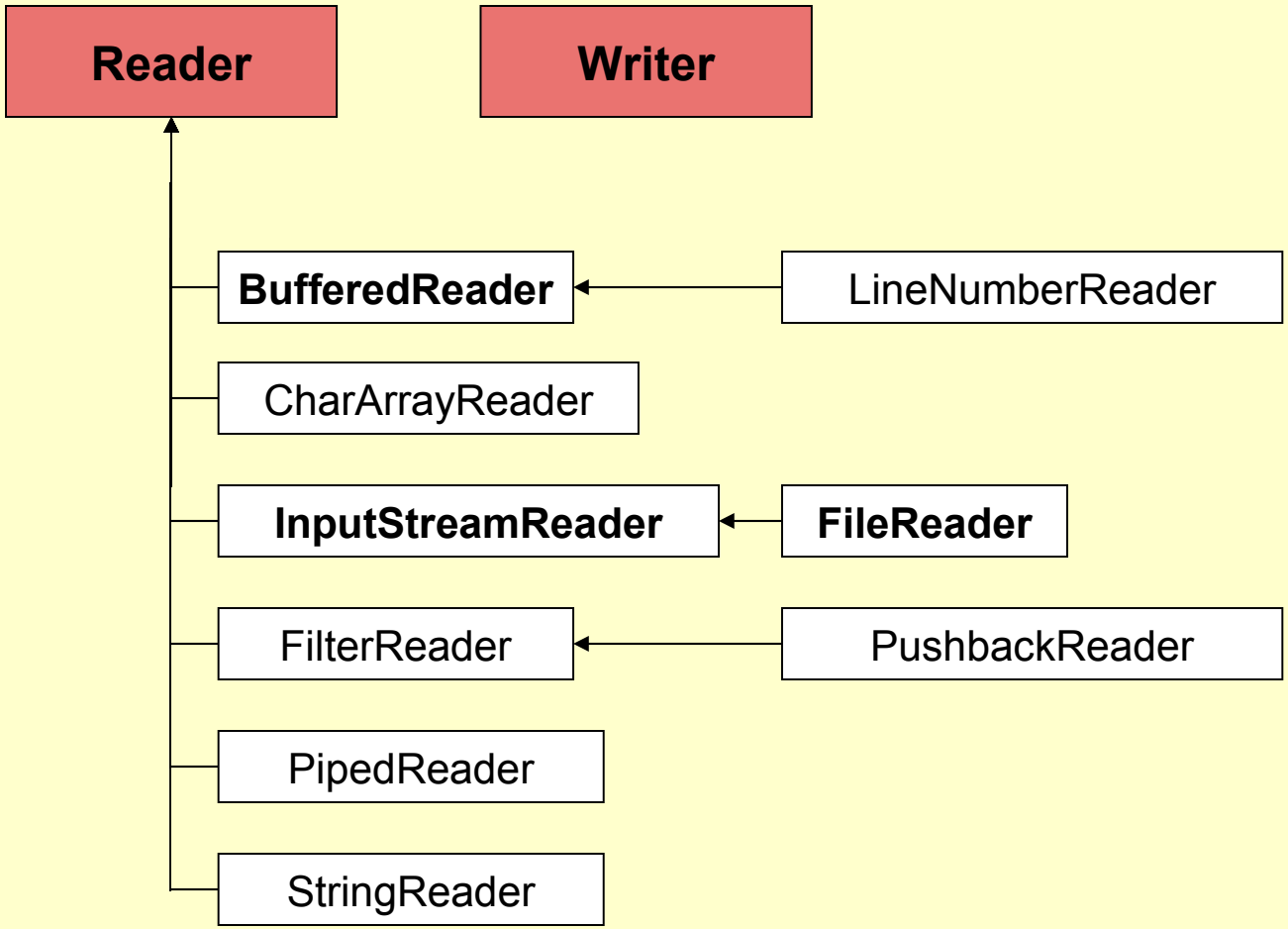

- **Karakter stream**:
  - 16 bites, UNICODE karaktereket tartalmaz („text”)
  - Írás, olvasás: **Reader** és **Writer**
  - pl. text file, xml, html
- **Byte stream**:
  - 8 bites, azaz 1 byteos adatokat tartalmaz („bináris”)
  - Írás, olvasás: **InputStream**, **OutputStream**
  - pl. video, audio, bináris adatfile

# API

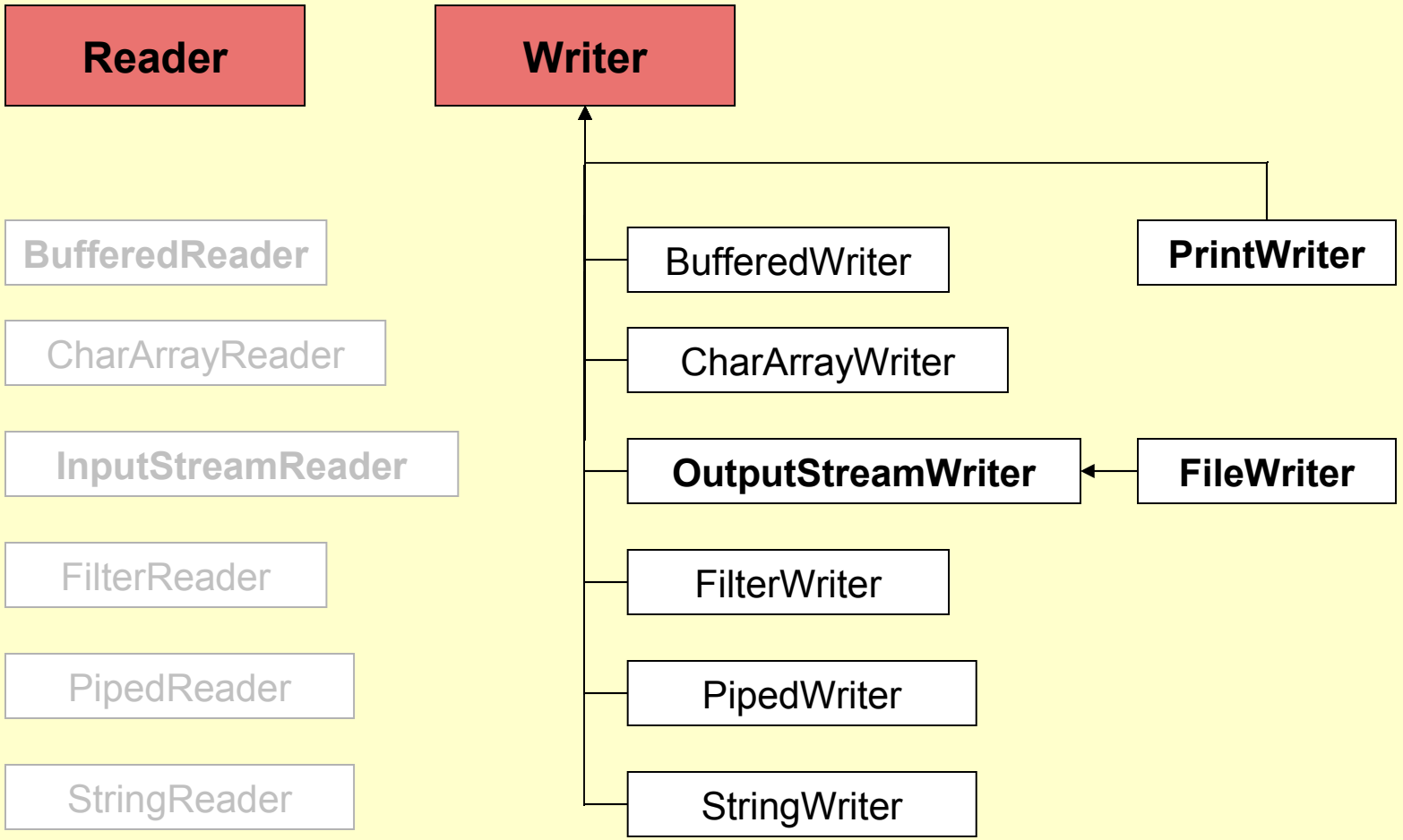
| karakter stream | byte stream |
|---|---|
| `int read()`<br>`int read(char cbuf[])`<br>`int read(char cbuf[],`<br>`  int offset, int length)` | `int read()`<br>`int read(byte cbuf[])`<br>`int read(byte cbuf[],`<br>`  int offset, int length)` |
| `void write(int c)`<br>`void write(char cbuf[])`<br>`void write(char cbuf[],`<br>`  int offset, int length)` | `void write(int c)`<br>`void write(byte cbuf[])`<br>`void write(byte cbuf[],`<br>`  int offset, int length)` |

**`throws IOException`**

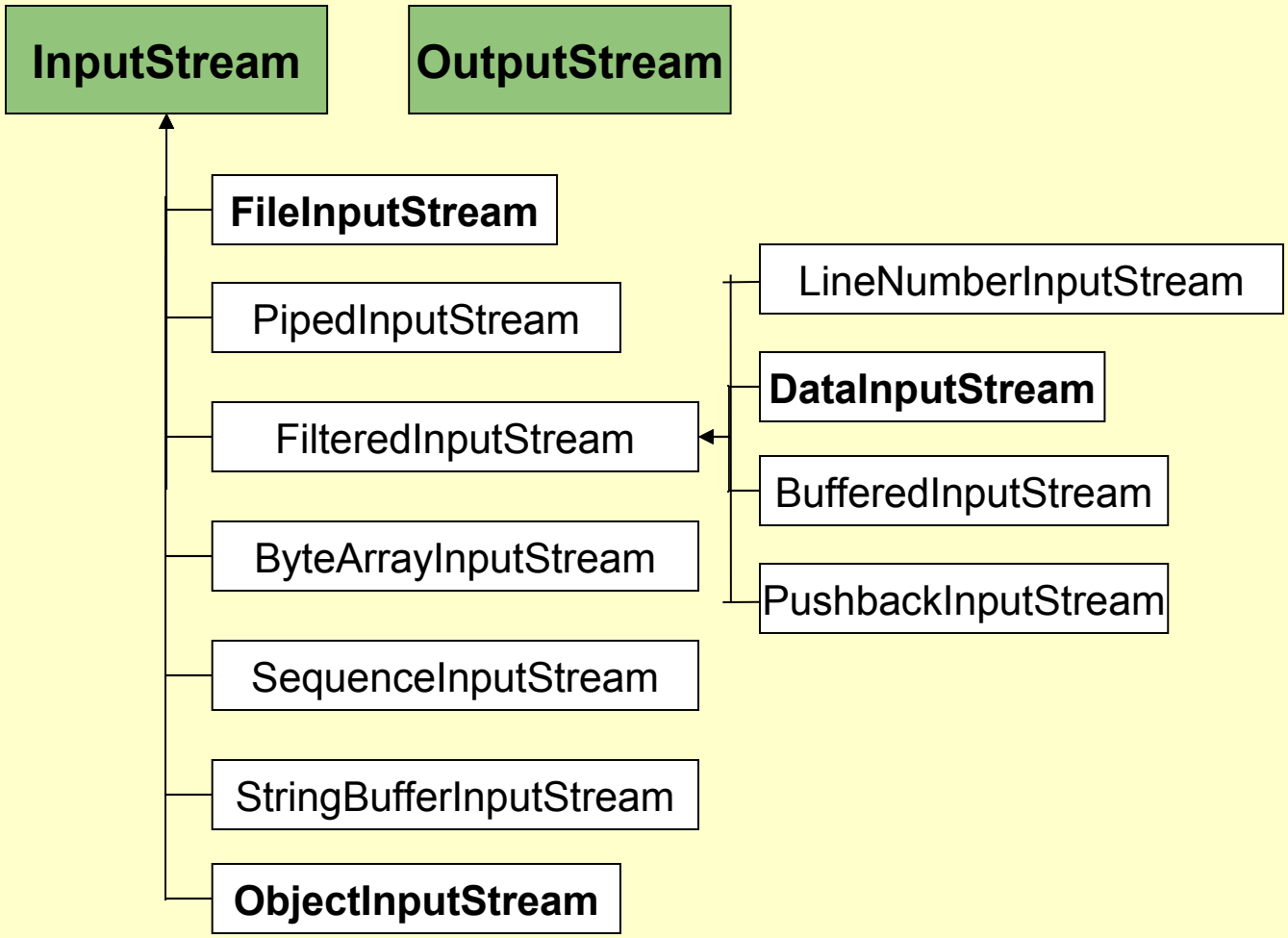
# Karakter stream



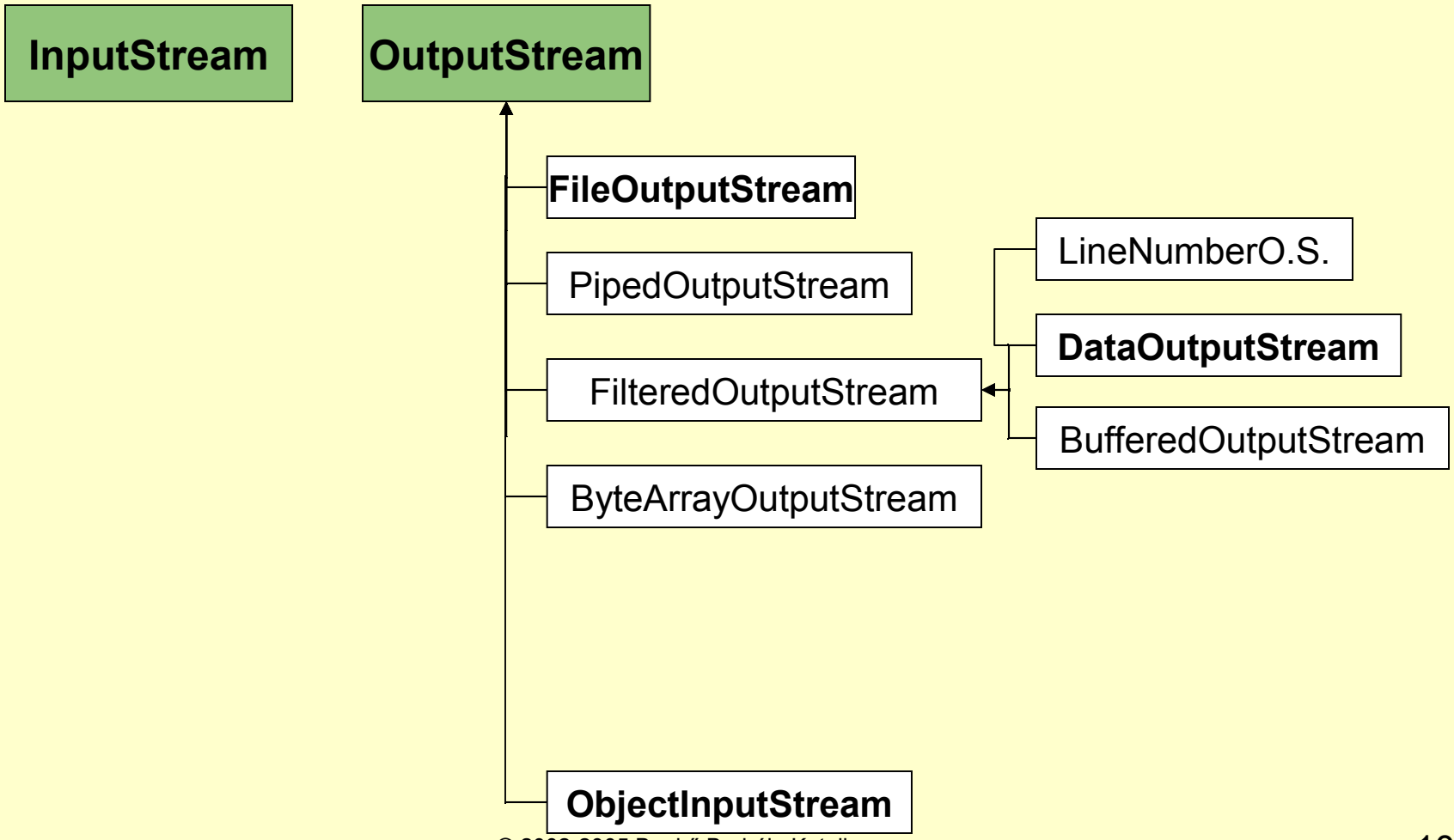
# Karakter stream



# Byte stream



# Byte stream



# Áttekintés

- Az adat ki/bevitel streambe való írást/olvasást jelent
- A stream kezelése: megnyitás, írás/olvasás, bezárás
- Megkülönböztetünk karakteres és byte-os streameket
- A különböző típusú streamek kezelésére számos io osztály áll a rendelkezésünkre

- A stream vége végnek elérését egyes osztályokban speciális visszatérési érték máshol Exception.

# Tematika

- Honnan olvashatunk be: (hely?)
  - File (karakteres/bináris)
  - Csövek
  - Memória
  - Konverzió
- Hogyan olvassunk be kényelmesen: (milyen egységet?)
  - A „szűrő” fogalma
  - Típusos IO
  - Bufferelés
  - Egyéb lehetőségek
  - Objektumos IO

# File kezelés

- Honnan olvashatunk be:
  - **File karakteres/bináris**
  - Csövek
  - Memória
  - Konverzió

# File io

- A csatornához a filerendszer egy file-ját rendeltük
- A konstruktornak megadhatunk
    - A file nevét Stringként (relatív vagy abszolút)
    - File objektumot: fájl vagy mappa
        - A mappa tudja listázni a tartalmát
        - Metaadatok lekérdezése
    - FileDescriptort
- Hiba: FileNotFoundException (IOException leszármazottja, ellenörzött)
- Jogosultság hiánya: SecurityException (nem ellenörzött)

# Karakteres fájl

- Ha az IO forrása vagy célja karakteres fájl, a **FileReader**, illetve **FileWriter** osztályt használhatjuk.
- Példa: karakteres fájl másolása.

```
import java.io.*;

public class TextCopy{
    public static void main(String [] args) {
        FileReader in = new FileReader("in.txt");
        FileWriter out = new FileWriter("out.txt");



        in.close();
        out.close();
    } // main vege
} // class vege
```

# Karakteres fájl

```
import java.io.*;

public class TextCopy{
    public static void main(String [] args) {
        FileReader in = new FileReader("in.txt");
        FileWriter out = new FileWriter("out.txt");

        while(true) {
            char c = in.read();
            if (c == -1) break;
            out.write(c);
        }

        in.close();
        out.close();
    } // main vege
} // class vege
```

# Karakteres fájl

```
import java.io.*;

public class TextCopy{
    public static void main(String [] args) {
     try {
        FileReader in = new FileReader("in.txt");
        FileWriter out = new FileWriter("out.txt");

        while(true) {
            char c = in.read();
            if (c == -1) break;
            out.write(c);
        }

        in.close();
        out.close();
     } catch (IOException e){
            System.out.println(e);
     }
    } // main vege
} // class vege
```

# Bináris fájl

- Ha az io forrása vagy célja bináris file, a **FileInputStream** illetve **FileOutputStream** osztályokat alkalmazhatjuk.
- Feladat: cseréljük ki az in.bmp fehér pixeleit pirosra!

```
import java.io.*;

public class BmpRed{
    public static void main(String [] args) {
        FileInputStream in = new FileInputStream("in.bmp");
        FileOutputStream out = new FileOutputStream("out.bmp");


        in.close();
        out.close();
    } // main vege
} // class vege
```

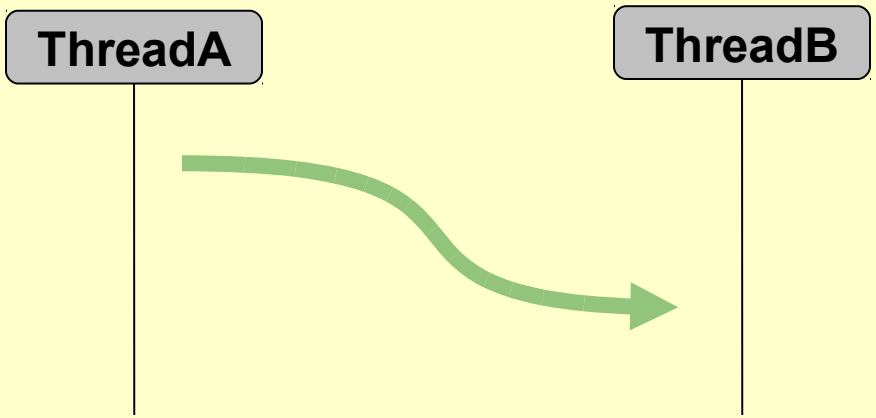
```
import java.io.*;

public class BmpRed {
    public static void main(String [] args) {
   try{
        FileInputStream in = new FileInputStream("in.bmp");
        FileOutputStream out = new FileOutputStream("out.bmp");

        byte header[] = new byte[108];
        in.read(header);
        out.write(header);
        while(true){
            int pix = in.read();
            if (pix == -1) break;
            if (pix == 0) pix = 3;
            out.write(pix);
        }
        in.close();
        out.close();
    } catch (IOException e) {
     System.err.print(e);
    }
    } // main vege
} // class vege
```

# Csövek

- Honnan olvashatunk be:
  - File karakteres/bináris
  - **Csövek**
  - Memória
  - Konverzió

# Csövek

- A UNIX pipe-hoz hasonló, csatornák egymásba kapcsolására.
- Fontos: szálak közti kommunikáció során
- Lehet, hogy a cső eleje és vége külön szálon van. A pipe „úgymond” a közvetlen függvényhívást helyettesíti.



# Csövek

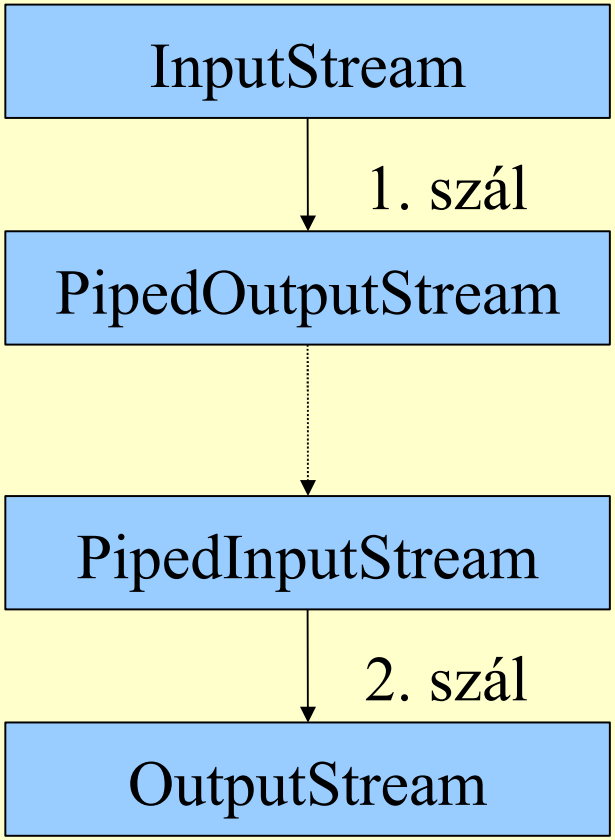
- Az összekötött csatornák lehetnek:
  - PipedReader és PipedWriter
  - PipedInputStream és PipedOutputStream
- Az összekötés menete:

```
PipedInputStream in = new PipedInputStream();
// extra code
in.connect(out);
```

```
PipedInputStream in = new PipedInputStream(out);
```

- Blokkolva olvasás

# Csövek - Példa




```
InputStream in = new FileInputStream(”in.txt”);

PipedOutputStream pout = new PipedOutputStream();
PipedInputStream pin = new PipedInputStream(pout);

OutputStream out = new FileOutputStream(”out.txt”);

Thread t1 = new CopyThread(in, pout);
Thread t2 = new CopyThread(pin, out);
t1.start();
t2.start();
```

# Memória

- Honnan olvashatunk be:
  - File karakteres/bináris
  - Csövek
  - **Memória**
  - Konverzió

# Memória

- Ha a forrás/cél adat memóriabeli tömbben van, használjuk a
  - ByteArrayReader / ByteArrayWriter –t
  - CharArrayReader / CharArrayWriter –t
  - StringReader/StringWriter -t
- Tipikus ilyen eset: amikor pl. C-ben temp fájlokat használnánk
- A konstruktorukban kapott tömböt/tömbrészletet olvassák végig vagy írnak bele.
- Nézzük a részletes tudnivalókat a beolvasásról és a kiírásról.

# Memória - olvasás

- A `CharArrayReader` nem készít másolatot a tömbről!

```
char [] t = {'f', 'a', 'l'}
CharArrayReader in = new CharArrayRader(t);
t[0] = 'h';
System.out.println( in.read() );  // → h
```

- A `ByteArrayInputStream` és `StringReader` tartalma megnyitás után nem módosítható (immutable).

# Memória - kiírás

- Működés:
  - Ő hozza létre a tömböt/Stringet stb, amibe ír.
  - A végén a tartalom lekérdezhető tőle.

  - Belső algoritmusa:
    - létrehoz egy tömböt, amibe ír
    - ha ez kicsinek bizonyul, megnöveli a tömb méretét
  - Megadható a buffer kezdeti hossza és a növekedés mértéke.
  - Default:növekedés = `2*x`

# Konverzió byte-os és karakteres csatornák között

- Honnan olvashatunk be:
  - File karakteres/bináris
  - Csövek
  - Memória
  - **Konverzió**

# Konverzió byte és karakter stream között

- Byte csatornához karakteres felületet biztosít: `InputStreamReader` és `OutputStreamWriter`
- A byte-ok alapértelmezésben Unicode-dá konvertálódnak, de megadható a kódolás típusa.
- A `FileReader` és `FileWriter` a leszármazottaik, de sajnos nincs nekik megfelelő konstruktoruk a kódolás megadásához.
- Példa: beolvasás a billentyűzetről, 8859_2 karakterkódolással:

```
InputStreamReader in = new InputStreamReader(System.in, “8859_2”);
```

# Szűrők

- Hogyan olvassunk be kényelmesen:
  - **Szűrők**
  - Típusos IO
  - Bufferelés
  - Egyéb lehetőségek
  - Objektumos IO

# Szűrők

- Azokat az osztályokat, amelyek nem az io *honnan*-ját hanem az *olvasás/írás egységét* tárgyalják, szűrőnek nevezzük.
- A szűrők mindig egy *honnant* tárgyaló streamet csomagolnak be

```
FileInputStream fin = new FileInputStream("vadalma.in");
SZŰRŐ in = new SZŰRŐ(fin);
```

```
SZŰRŐ in = new SZŰRŐ(new FileInputStream("vadalma.in"));
```

- **Decorator design pattern**
- A legfontosabb szűrők:
  - DataInputStream / DataOutputStream
  - BufferedReader / PrintWriter
  - ObjctInputStream / ObjectOutputStream

# Típusos IO

- Hogyan olvassunk be kényelmesen:
  - Szűrők
  - **Típusos IO**
  - Bufferelés
  - Egyéb lehetőségek
  - Objektumos IO

# Típusos IO

- A **DataInputStream / DataOutputStream** segítségével a primitív adattípusokat tudjuk egyszerűen beolvasni és kiírni.
- Minden adattípushoz külön író/olvasó függvény van.
- Példa: primitív adattípusok fájlba írása

```
DataOutputStream out = new DataOutputStream(new
                                    FileOutputStream("bin.b"));
for (int i = 0; i < 10; i++){
    out.writeDouble(i + 0.5);
    out.writeChar('\t');
    out.writeInt(i);
}
out.close();
```

# Típusos IO

- Példa: beolvasás

```
DataInputStream in = new DataInputStream(new
                 FileInputStream("bin.b"));
try {
    while(true) {
    double d = in.readDouble();
    in.readChar();
    int i = in.readInt();
    }
}catch (EOFException e) {
    in.close();
}
```

- Az olvasás végét Exception jelezte.

# Bufferelt olvasás

- Hogyan olvassunk be kényelmesen:
  - Szűrők
  - Típusos IO
  - **Bufferelés**
  - Egyéb lehetőségek
  - Objektumos IO

# Bufferelt olvasás

- Praktikus, ha egy fájl tartalmát nem csak karakterenként vagy byte-onként érhetjük el, hanem egy nagyobb szakaszt egyben is.
- A bufferelt olvasás lényege, hogy a stream egy szakaszát egy bufferbe olvassa
  - De nem kell előre tudnom, hogy pontosan mennyi adatot, hanem csak azt, hogy pl. milyen karakterig
- Példa: szöveges fájlok kezelése – soronkénti olvasás
- Osztályok:
  - `BufferedReader`, `BufferedWriter` / `PrintWriter`
  - `BufferedInputStream`, `BufferedOutputStream` / `PrintStream`

# Szövegfeldolgozás

- **BufferedReader**
  - Képes egy sort egy művelettel beolvasni.
  - A BufferedReader egy bufferbe olvas
    - Ha a buffer túlcsordulna, megduplázódik
    - A buffer kezdeti mérete megadható.
  - Lényege: a buffer tartalma egy művelettel elérhető
  - Byte-os párja: BufferedInputStream

```
BufferReader in = new BuffereReader(
                                    new FileReader(“in.txt”));
String line = in.readline();
in.close();
```

# Szövegfeldolgozás

**PrintWriter**

- A kiírandó adatot karakterenként írja a kimeneti streamre.
- A kiírandó objektum toString metódusát hívja meg.
- A sorvége jelet automatikusan lecseréli az adott platformon használatosra.
- `autoflush` megadható
- Byte-os párja: PrintStream.

```
PrintWriter out = new PrintWriter(
                                  new FileWriter("out.txt"));
out.println("vadkörte");
out.close();
```

- Példa: szövegfile feldolgozása

# Példa

- Szövegfájl feldolgozása
  - Soronként egy név és számok vannak
  - Számítsuk ki a nevenkénti összeget
  - Írjuk ki egy másik fájlba

**in.txt**

```
Törpapa 3.0 4.5 2.0
Törpilla 5.0 2.1
Okoska 13.0
```

**out.txt**

```
Törpapa 9.5
Törpilla 7.1
Okoska 13.0
```

# Példa

```
try {
  BufferedReader in = new BufferedReader(new FileReader("in.txt"));
  PrintWriter out = new PrintWriter(new FileWriter("out.txt"));
  for ( String line = in.readLine(); line != null; line = in.readLine() ) {
   double sum = 0;
   StringTokenizer st = new StringTokenizer(line);
    String name = st.nextToken();
   while ( st.hasMoreTokens() ) {
      String d = st.nextToken();
      sum += Double.parseDouble(d);
   }
   out.println(name + " " + sum);
  }
  in.close();
  out.close();
} catch(IOException e){
  System.err.println(e);
}
```

line.split() is jó lenne a StringTokenizer helyett

# Standard IO

- Hogyan olvassunk be kényelmesen:
  - Szűrők
  - Típusos IO
  - Bufferelés
  - **Egyéb lehetőségek**
  - Objektumos IO

# Standard IO

- System.in
  - InputStream

```
BufferedReader in = new BufferedReader(
   new InputStreamReader(System.in));
```

- System.out, System.err
  - PrintStream – soha nem dob Exception-t

```
PrintWriter out = new PrintWriter(
   new OutputStreamWriter(System.out));
```

- Persze ez valószínűleg felesleges, mert println a PrintStream-ben is van.

- System.console() → Reader/PrintWriter pár, 1.6 óta, nem szabható testre

# Encoding 1.

- Az esetek döntő többségében jó a default encoding detection.
- Lehet explicit megadni: encodingName vagy decoder.
  - charsetname (pl US-ASCII, ISO-8859-1, UTF-8, UTF-16BE, UTF-16LE, UTF-16)
  - `Charset`
  - `CharsetDecoder`
    - Egy byte tömböt alakít karakter tömbbé.
    - Így tetszőleges méretű karakterek kezelése megoldható.

# Encoding 2.

- Csak az InputStreamReader / OutputStreamWriter –nek adható meg kódolás, a leszármzaottaknak nem. Ezért pl. fájlból való kódolt olvasáshoz is többszörös szűrésre (becsomagolára) van szükség.

```
static final String encoding = "ISO-8859-2";
BufferedReader in = new BufferedReader(
        (Reader)(
            new InputStreamReader(
                new FileInputStream(" in.txt"),
                encoding)
            )
        );
```

# Encoding 3.

```
PrintWriter out = new PrintWriter(
        new OutputStreamWriter(
            new FileOutputStream(”out.txt”), encoding )
        )
);
```

```
static final String encoding = "ISO-8859-2";
BufferedReader in = new BufferedReader(
        new InputStreamReader( System.in, encoding)
);
```

# Ami kimaradt

# Ami kimaradt

- Byte-os csatornák egymás után fűzhetők a **SequenceInputStream** segítségével

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

- A **LineNumberInputStream** egy szűrő, amely nyilvántartja az olvasott sor sorszámát.
- **PushbackReader**
- **FilterReader**
- **RandomAccessFile**

# StreamTokenizer

- Szövegfeldolgozásra való, hasonlít a StringTokenizer-hez
- Karakteres csatorna felett
- A konstruktor Reader-ben kapott szöveget darabolja az elválasztó karakterek mentén
  - int nextToken()
  - nval, sval

```
StreamTokenizer st = new StreamTokenizer(
    new FileReader("in.txt"))
int m = st.countTokens();

for (int i = 0; i < m; i++) {
    if (st.nextToken() == StreamTokenizer.TT_NUMBER)
        System.out.println(st.nval)
}
```

# Objektumos IO

- Hogyan olvassunk be kényelmesen:
  - Szűrők
  - Típusos IO
  - Bufferelés
  - Egyéb lehetőségek
  - **Objektumos IO**

# Objektumos IO

- Lehetőség van objektumok közvetlen kiírására és beolvasására
- Osztályok: `ObjectInputStream`, `ObjectOutputStream`
- Kapcsolódó fogalom: szerializáció
- Az objektumos io is szűrő

```
FileOutputStream fOut = new FileOutputStream("out.bin");
ObjectOutputStream oOut = new ObjectOutputStream(fOut);
...
oOut.flush();
fOut.close();
```

- Példa: objektumok kiírása és beolvasása

# Objektumos kiírás

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

class ObjectW {
public static void main(String [] args) {
    try {
        FileOutputStream fOut =
            new FileOutputStream(”obj.bin”);
                ObjectOutputStream oOut =
                    new ObjectOutputStream(fOut);
        for (int i = 0; i < 10; i++){
        oOut.writeObject(new String(”aaa” + i + ”bbb”));
        oOut.writeObject(new KakaósPalacsinta());
                }
        oOut.close();
    } catch (IOException e) {
       System.err.print(e);
    }
}
}
```

# Objektumos beolvasás

- **Beolvasás:**

```
FileInputStream fIn = new FileInputStream(”obj.bin”);
ObjectInputStream oIn = new ObjectInputStream(fIn);
try {
    while(true) {
        String s = (String) oIn.readObject();
        Date d = (KakaósPalacsinta) oIn.readObject();
    }
} catch (EOFException e) {
    oIn.close(); // + try-catch kellene köré
} catch (FileNotFoundException e) {
    System.err.print(e);
} catch (IOException e) {
    System.err.print(e);
} catch (ClassNotFoundException e) {
    System.err.print(e);
}
```

# Szerializáció

- A szerializáció egy objektum byte-sorozattá alakítását jelenti. Az objektum és a bytefolyam közti leképezés kölcsönösen egyértelmű.
- Csak az az osztály szerializálható, amely megvalósítja a `Serializable` interface-t.
- **interface Serializable**

```
package java.io;
public interface Serializable {
    // there's nothing in here!
};
```

- Általában: automatikus szerializáció
- Lehetőség van saját szerializáció definiálására

# Automatikus szerializáció 1.

- Automatikus metódusok:

```
ObjectOutputStream.defaultWriteObject()
    throws IOException;
ObjectInputStream.defaultReadObject()
    throws IOException, ClassNotFoundException;
```

- **Az automatikus szerializációkor kiíródik:**
  - Az objektum osztálya, az osztály signature-je.
  - Az objektum nem-statikus és nem-tranziens mezőinek értéke.
  - Az objektum referenciáinak tranzitív lezártja (azaz minden, amire bármilyen, akár közvetett úton van referenciája).
- Tranziens mező: **transient** módósítószó → nem íródik ki a szerializációkor

# Automatikus szerializáció 2.

- **Legfontosabb szabályok:**
- Csak az az osztály szerializálható, amely megvalósítja a `Serializable` interface-t.

```
class MySerializableClass implements Serializable {
    ...
```

- Ha egy osztály szerializálható, a leszármazottjai is azok.
- Az objektummal együtt referenciáinak tranzitív lezártja is kiírásra kerül. Itt a referenciális integritás elve érvényesül.
- Ha egy osztály nem szerializálható, NotSerializableException-t dob. A hiba futási idejű.

# Szerializáció korlátozása

- Esetenként szükséges a szerializáció felülbírálata.
  - „titkosan”  kezelendő adatok
  - sávszélesség limit (pl. mobiltelefonos környezet vagy nagyon sok kliens)
- Lehetőségek:
  - Az osztály nem valósítja meg a Serializable interface-t, nem objektumos IO-t választunk.
  - A „titkos” / nem fontos mezőt tranzienssé tesszük.
  - Felüldefiniáljuk a szerializációt.

# Szerializáció felüldefiniálása

- fel kell venni egy readObject és egy writeObject metódust

```
private void writeObject(ObjectoutputStream out)
    throws IOException;
private void readObject(ObjectInputStream in)
    throws IOEXception, ClassNotFoundException;
```

- ezek a függvények kezdődhetnek (ajánlottan kezdődnek is!) a default függvényekre való hivatkozással

# Szerializáció felüldefiniálása

```
class MySerializableClass implements Serializable {
    ...
    protected transient int titok;
    ...

    private void writeObject(ObjectOutputStream out) throws
IOException {
        out.defaultWriteObject();
        out.writeInt( encrypt(titok, getKey()) );
    }

    private void readObject(ObjectOutputStream in) throws
    IOException, ClassNotFoundException {
        in.defaultReadObject();
        titok = decrypt( in.readInt(), getKey() );
    }
}
```

# Externalizáció

- A Serializable felüldefiniálása néha nem elég, mert private.

```
public Intreface Extrernalizable extends Serializable {
    public void writeExtrenal(ObjectOutput out)
        throws IOException;
    public void readExtrernal(ObjectInput in)
        throws IOException, ClassNotFoundException;
}
```

- Az osztálynak teljes kontrollja van a kiírás felett:
  - Csak az osztályazonosító íródik ki automatikusan
  - Saját felelősség meghívni az ősöket
  - Visszatöltés menete: az osztály default (no-arg) konstruktora hívódik meg, majd a readExtrenal

# Verziók

- Ha egy objektumot szerializáltunk, majd az osztályát módosítjuk, a változás lehet kompatibilis vagy nem kompatibilis.
- Az új verzió felelőssége, hogy a régit jól olvassa be.
- Kompatibilis változások
  * mezők jogosultság-változása (public, protected, private)
  - új mezők felvétele (default érték)
  - statikus → nem statikus, tranziens → nem-tranziens
  - új osztályok bevétele vagy elhagyása az öröklési láncban (default érték ill. eldob)
  - új readObject vagy writeObject létrehozása vagy régiek elhagyása (feltéve, hogy defaultRead/Write/Object-tel indul)

# Verziók

- Nem kompatibilis változások:
  - mező elhagyása
  - nem statikus → statikus, nem tranziens → tranziens
  - öröklődési hierarchia felborítása (az eredeti láncból való kilépés)
  - primitív adattípus mező megváltoztatása
  - writeObject, readObject megváltoztatása úgy, hogy az egyikben a default a másikban más értékeket írnak ki
  - Serializable-ről Externalizable-re való áttérés

- protected static final long SERIAL_VERSION_UID = 20130305L;

# IO típusok összefoglalás

- Alapfogalom: stream
- A stream lehet karakteres vagy byte-os: (pl.File)
  - Karakteres: `FileReader/Writer`
  - Byte-os: `FileInputStream` / `FileOutputStream`
- Szűrőkkel kényelmesebbé tehetjük a stream használatát:
  - `DataInput/Output/Stream`
  - `ObjectInput/OutputStream`
  - `BufferedReader`, `PrintWriter`

# IO típusok összefoglalás

- Szerializáció
  - Automatikus
  - Serializable felüldefiniálása
  - Extrernalizable

- Amiről nem volt szó: random hozzáférésű file-ok
  - RandomAccessFile osztály
  - Jar (zip) kezelés