Java programozási nyelv 2012-2013/ősz
4. óra

TCP/IP kapcsolat Java nyelven

Java streamek, szűrők, java.io
TCP/IP alapú kommunikáció
Egyszerű protokoll tervezése

legradi.gabor@nik.uni-obuda.hu szenasi.sandor@nik.uni-obuda.hu

TCP/IP kapcsolat Java nyelven

Témakörök

Streamek kezelése Javában

Szűrők és használatuk

TCP/IP kapcsolat felépítése

TCP/IP a szerver oldalon

Kliens/szerver alkalmazás megvalósítása

Streamek kezelése Javában

- Streamek: adatfolyamok, amelyeken keresztül lehetőségünk van adatokat írni/olvasni
- A *java.io* csomag tartalmazza a szükséges osztályokat
- A streamek használata hasonló a C#-ban tanultakhoz
- Egy stream életciklusa:
  - **Megnyitás**
    A megfelelő stream-osztály példányosításával. (Általában a konstruktornak átadott paraméterek segítségével jelöljük ki az elérni szánt médiát)
  - **Írás/olvasás**

    A stream megfelelő metódusai segítségével
  - **Bezárás**

    A stream objektum close() metódusával

Streamszervezési módok

A streamek őseik alapján két csoportba oszthatók

- **Byte szervezésű streamek**
  (*InputStream/OutputStream* leszármazottai)
  A legkisebb egység a byte. Ezeket tudjuk írni/olvasni a csatornán keresztül
- **Karakter szervezésű streamek**
  (*Reader/Writer* leszármazottai)
  A legkisebb egység a Unicode karakter. Ezeket tudjuk írni/olvasni a csatornán keresztül.
  Fontos különbség, hogy a karakter szervezésű streamek működésük során figyelembe veszik a különböző kódolások sajátosságait

Kommunikáció streameken keresztül

- Az absztrakt ősosztályok definiálják az alapvető írási műveleteket
- Byte szervezésű stream (*OutputStream*)
  - write(int a) throws IOException
  - write(byte b[]) throws IOException
  - write(byte b[], int off, int len) throws IOException
- Karakter szervezésű stream (*Writer*)
  - write(int c) throws IOException
  - write(char c[]) throws IOException
  - write(char c[], int off, int len) throws IOException
  - write(String s)
- Puffer ürítése
  - A flush() metódus segítségével
  - Stream lezárásakor automatikusan meghívódik

Kommunikáció streameken keresztül

- Byte szervezésű stream (*InputStream*)
  - int read() throws IOException
  - int read(byte c[]) throws IOException
  - int read(byte c[], int off, int len) throws IOException
- Karakter szervezésű stream (*Reader*)
  - int read() throws IOException
  - int read(char c[]) throws IOException
  - int read(char c[], int off, int len) throws IOException

Visszatérési érték a beolvasott elemek száma

Állapot	Visszatérési érték
Új adat érkezett	A beérkező szám/karakter kódja
Adatfolyam vége	-1
Nincs adat	A stream blokkolódik, várakozik a következő küldeményig

Kommunikáció streameken keresztül

Példa – stream másolása

```
public static void masol(InputStream in, OutputStream out) {
        try {
                int b;
                while (b = in.read() != -1) out.write(b);
                out.flush();
        } catch (IOException e) {
                System.out.println(e.getMessage());
        }
}
```

Leképezés fizikai adatszerkezetre

- Fájlokra leképező osztályok
  - Karakter: *FileReader/FileWriter*
  - Byte: *FileInputReader/FileOutputStream*
- Lehetséges konstruktorok
  - FileReader(String filenév) throws FileNotFoundException
  - FileReader(File file) throws FileNotFoundException
  - FileReader(FileDescriptor fd)
- Csövekre leképező osztályok
  - *PipedInputStream/PipedOutputStream*
  - *PipedReader/PipedWriter*
- Tömbökre leképező osztályok
  - pl. *CharArrayWriter* stb.

TCP/IP kapcsolat Java nyelven

Témakörök

Streamek kezelése Javában

Szűrők és használatuk

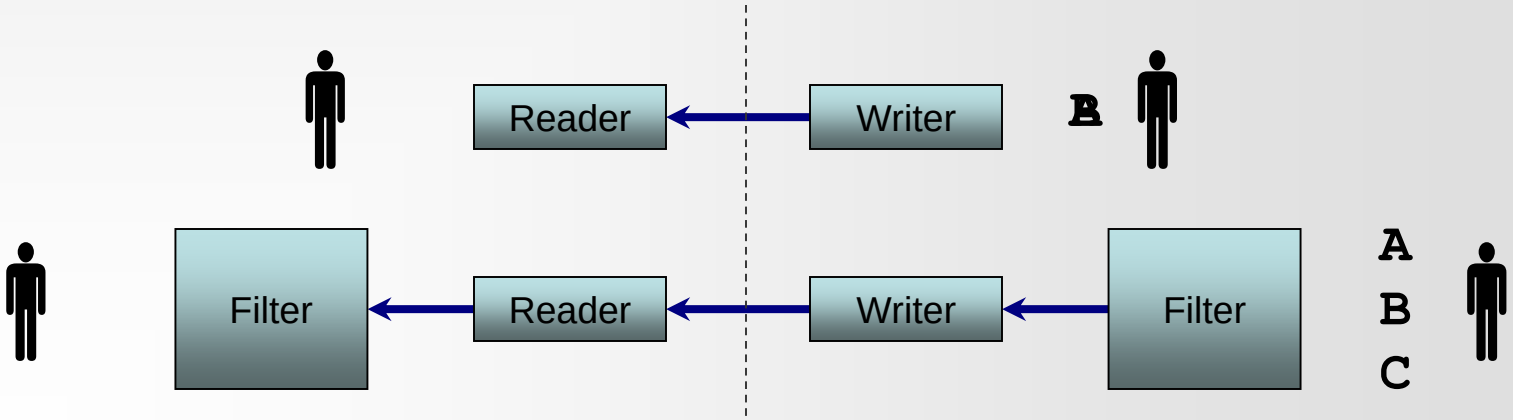
TCP/IP kapcsolat felépítése

TCP/IP a szerver oldalon

Kliens/szerver alkalmazás megvalósítása

Szűrők alapvető szerepe

- A szűrők nem közvetlenül egy fizikai médiumhoz kapcsolódnak, hanem egy másik streamhez, ezzel segítik elő az adatfolyam magasabb szintű kezelhetőségét
- A szűrők is a stream osztályok leszármazottai, ezért ők is csatornaként használhatók
  (így akár egymás után több szűrő is kapcsolható)



Szűrők használata

- Szűrő objektumok létrehozásakor általában a konstruktorban kell átadni azt a csatornát, amelyik felett szeretnénk a szűrőt használni
  FileOutputStream fout = new FileOutputStream("...");
  DataOutputStream dos = new DataOutputStream(fout);
- A szűrők egymásba is ágyazhatók (delegáció), de egy csatornához közvetlenül ne rendeljünk több szűrőt (amennyiben mégis, a flush()-el nekünk kell elkerülni a pufferelésből adódó problémákat)
- Hasonló okból használat során már csak a szűrőn keresztül férjünk hozzá a csatornához

  dos.writeFloat(...);
- Lezáráskor elég a szűrőt lezárni, ez elvégzi a csatorna lezárását is
  dos.close();

Java alaptípusok írása/olvasása

- A *DataOutputStream* szűrővel van lehetőség a Java alaptípusok írására **byte** típusú csatornára
  - public void writeInt(int v) throws IOException
  - public void writeFloat(float v) throws IOException
  - public void writeChar(char v) throws IOException
  - ...
- A *DataInputStream* szűrővel van lehetőség az így kiírt alaptípusok beolvasására egy **byte** szervezésű csatornából
  - public int readInt() throws IOException
  - public float readFloat() throws IOException
  - public char readChar() throws IOException
  - ...

Szöveges formában írás

- A *PrintWriter* szűrővel van lehetőség egy **karakteres** csatornára szöveges formában írni a Java alaptípusokat Kiírás után soremeléssel:
  - public void println(int v) throws IOException
  - public void println(float v) throws IOException
  - public void println(String v) throws IOException
  - ...

  Soremelés nélkül:
  - public void print(int v) throws IOException
  - public void print(float v) throws IOException
  - public void print(String v) throws IOException
  - ...

Szöveges formában olvasás

- A *PrintWriter*-nek nincs „párja”. Az általa írt adatokat célszerű a *BufferedReader* szűrővel feldolgozni (ami **karakter** szervezésű csatornákon tud csak dolgozni)
  - public char read () throws IOException
    Egy karakter beolvasása
  - public string readLine() throws IOException
    Egy sor beolvasása

- Egy *LineNumberReader* szűrő közbeiktatásával számolhatjuk a bemeneti **karakter** szervezésű csatorna sorait is
  - public int getLineNumber()
    Olvasás közben számolja, hogy hányadik sorban jár, ezt a számot adja vissza ez a metódus

További szűrők

- Bytecsatorna feletti karaktercsatorna
  InputStreamReader/OutputStreamWriter
  Konstruktoruk egy byte szervezésű stream objektumot vár, ők pedig karakteres streamként kezelhetők. Így tudunk byte szervezésű adatfolyamot karaktereskénti kezelni. Ha szükséges, a konstruktorban van lehetőség kódtábla megadására is.
- Tömörítés
  ZipInputStream/ZipOutputStream
  GZIPInputStream/GZIPOutputStream
- Ellenőrzött átvitel
  CheckedInputStream/CheckedOutputStream
- Titkosítás
  CipherInputStream/CipherOutputStream
- Objektumok továbbítása
  ObjectInputStream/ObjectOutputStream

TCP/IP kapcsolat Java nyelven

Témakörök

Streamek kezelése Javában

Szűrők és használatuk

TCP/IP kapcsolat felépítése

TCP/IP a szerver oldalon

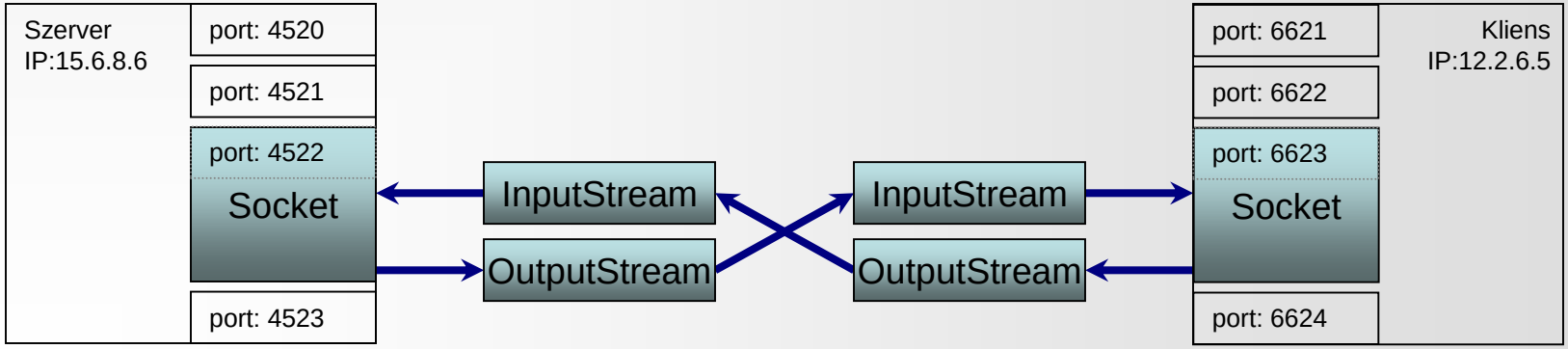
Kliens/szerver alkalmazás megvalósítása

TCP/IP kapcsolat

- TCP alapú kommunikáció socketeken keresztül zajlik. Egy socketet az IP cím és a portszám azonosít egyértelműen
- A szerver és a kliens csak a kapcsolat felvételének módjában különbözik egymástól (kliens kezdeményez, a szerver pedig fogadja a kapcsolatot)
- A kapcsolat felépítését követően a két fél már egyenrangúnak tekinthető, mindkét oldalon ugyanolyan socketek biztosítják a kommunikációt
- Ezt jól mutatja, hogy az implementáció során is ugyanazt a *Socket* osztályt használjuk a kliens és a szerver alkalmazás elkészítése során

TCP kapcsolat felépítése

- A *java.net* csomag tartalmazza a hálózati kapcsolathoz szükséges osztályokat
  - URL kezelés
  - TCP kapcsolat kezelése
  - UDP kapcsolat kezelése (nem foglalkozunk vele)
- A *Socket* osztály segítségével van lehetőség TCP alapú kapcsolat felépítésére
  A kapcsolat felépítését követően mindkét oldalon a létrejövő *Socket* objektumokon keresztül elérhető egy-egy kimeneti/bemeneti stream, ezeken keresztül lehet a kommunikációt lefolytatni



Socket osztály

- Kapcsolat felépítése a streamekhez hasonlóan a konstruktor meghívásakor történik az átadott paraméterek szerint
  - public Socket(String host, int port)
    throws UnknownHostException, IOException
    A host lehet ”192.168.0.5” vagy ”www.bmf.hu” alakú
    A portszám egy egész szám 1 és 65535 között
- A kapcsolat felépítését követően az alábbi metódusokkal lehet hozzáférni a csatornákhoz
  - public InputStream getInputStream() throws IOException
  - public OutputStream getOutputStream() throws IOException
- Kommunikáció és a csatornák lezárása
  - public void close() throws IOException

Socket osztály

- A létrehozott kapcsolat adatai lekérdezhetőek
  - public InetAddress getInetAddress()
    Távoli gép címe
  - public InetAddress getLocalAddress()
    Helyi gép címe
  - public int getPort()
    Távoli gépen nyitott port száma
  - public int getLocalPort()
    Helyi gépen nyitott port száma
- A kapcsolat állapotával kapcsolatos adatok
  - public boolean isConnected()
  - public boolean isClosed()
  - public boolean isInputShutdown()
  - public boolean isOutputShutdown()

Kliens oldali kommunikáció

Példa: kliens oldali kapcsolatfelvétel

```
try {
        Socket s = new Socket("ultra.obuda.kando.hu", 7);
        InputStream is = s.getInputStream();
        InputStreamReader isr = new InputStreamReader(is);
        BufferedReader br = new BufferedReader(isr);
        OutputStream os = s.getOutputStream();
        PrintWriter pw = new PrintWriter(os);
        pw.println("ECHO"); pw.flush();
        String answer = br.readLine(); System.out.println(answer);
        s.close();
} catch (Exception e) {
        System.out.println(e.getMessage());
}
```

TCP/IP kapcsolat Java nyelven

Témakörök

Streamek kezelése Javában

Szűrők és használatuk

TCP/IP kapcsolat felépítése

TCP/IP a szerver oldalon

Kliens/szerver alkalmazás megvalósítása

Szerver oldali implementáció

- A kapcsolatot mindig a kliens kezdeményezi, ezért a kliens oldali *Socket* objektum létrehozása egybeesik a kapcsolat kiépítésével
- A kapcsolat mindkét oldalán ugyanolyan *Socket* objektumokra van szükség, értelemszerűen ugyanabban az időpillanatban kell létrejönniük, ezt azonban meglehetősen nehéz lenne implementálni az eddigi eszközeinkkel
- Ezért rendelkezésre áll egy *ServerSocket* osztály, amelynek példányaival a szerver oldalon van lehetőségünk folyamatosan figyelni egy portot, hogy mikor érkezik be a következő kérés egy klienstől
- Amennyiben egy kliens csatlakozik ehhez a porthoz, akkor a *ServerSocket* objektum létrehoz egy *Socket* objektumot, amin keresztül maga a kommunikáció a már megismert módon végrehajtható

ServerSocket osztály

- Az *ServerSocket* objektum példányosításakor a konstruktorban kell átadni a figyelendő port számát
  - public ServerSocket(int port) throws IOException
    A *port* paraméter a figyelni kívánt port száma.
    Ha az átadott paraméter 0, akkor szabadon választ egyet az aktuálisan szabad portok közül. Ebben az esetben a szerver indítását követően a tulajdonságai közül lehet lekérdezni, hogy milyen porton indult el
  - public ServerSocket(int port, int backlog) throws IOException
    Amíg az utoljára beérkező kapcsolat kezelése miatt a *ServerSocket* objektum nem kapja vissza a vezérlést, egy várakozási sorba gyűjti a beérkező kapcsolódási kéréseket.
    A fentihez hasonló *port* utáni *backlog* paraméter határozza meg ennek a sornak a méretét. Az ebbe nem férő kéréseket elutasítja
- A *ServerSocket* objektum lezárása
  - public void close() throws IOException

ServerSocket figyelő aktiválása

- A példányosított *ServerSocket* objektum nem kezdi el azonnal a port figyelését. Erre szolgál az alábbi metódus
  - public Socket accept() throws IOException
- A metódus meghívása blokkolja a program futását egészen addig, amíg nem érkezik be egy kérés
- Kérés beérkezése esetén a metódus visszatérési értéke egy *Socket* objektum, amin keresztül hozzáférhetők a kliens adatai, illetve a kommunikációhoz szükséges streamek
- A beérkező kérés után a blokkolás megszűnik és a program fut tovább, a következő kliens kiszolgálásához újra meg kell hívni az accept() metódust.

Ezért célszerű

- Az accept() hívásokat és a kérések kiszolgálását ciklusba szervezni
- A kliensek kiszolgálását egy külön szálban elvégezni, hogy ezzel párhuzamosan a *ServerSocket* újra képes legyen figyelni a portot
- Az accept() hívásokat is egy háttérben futó szálon futtatni, ha nem szeretnénk emiatt blokkolni a teljes alkalmazásunkat

Szerver oldali kommunikáció

Példa: az echo protokoll megvalósítása

```
try {
        ServerSocket ss = new ServerSocket(7);
        while (true) {
                Socket s = ss.accept();
                InputStream is = s.getInputStream();
                InputStreamReader isr = new InputStreamReader(is);
                BufferedReader br = new BufferedReader(isr);
                OutputStream os = s.getOutputStream();
                PrintWriter bos = new PrintWriter(os);
                String question = br.readLine();
                bos.println(question); bos.flush();
                s.close();
        }
} catch (Exception e) {
        System.out.println(e.getMessage());
}
```

TCP/IP kapcsolat Java nyelven

Témakörök

Streamek kezelése Javában

Szűrők és használatuk

TCP/IP kapcsolat felépítése

TCP/IP a szerver oldalon

Kliens/szerver alkalmazás megvalósítása

Egyszerű protokoll tervezése

- A szerver és a kliens közötti kommunikáció tetszőleges lehet, de időben össze kell hangolni a kettő működését
- Értelemszerűen amit küld a kliens, azt olvasnia kell a szervernek. Amit pedig a szerver küld, annak a fogadására fel kell készülnie a kliensnek
- Egy egyszerű protokoll keretein belül célszerű rögzíteni az írások/olvasások
  - Időbeni sorrendjét
  - Irányát
  - Adatok típusát, értelmezési módját
- A protokoll megtervezése után jól elkülönülnek egymástól a kliens és a szerver feladatai, így ezeket már egymástól függetlenül is egyszerűen lehet implementálni
- Készítsünk egy kliens/szerver alkalmazást, ahol a kapcsolat kiépítését követően a kliens átküld a szervernek néhány stringet, majd a szerver ezek közül visszaküldi a kliens számára a leghosszabbat!

Esettanulmány

- A szervernek és a kliensnek kell-e azonosítaniuk egymást?
- Amennyiben a szerver egyszerre több szolgáltatást is nyújt ugyanazon a porton, akkor hogyan lehet közülük választani?
- A funkció tisztázását követően a kliensnek el kell küldenie a szerver számára a vizsgálandó stringeket.
- Honnan tudja a szerver, hogy hány stringet kell fogadnia?
  - pl. a kliens küldés előtt elküldi ezt a számot
  - pl. a kliens az utolsó string elküldése után elküld egy előre egyeztetett jelet, ezzel jelezve, hogy nem akar többet küldeni
- Milyen formában küldjük át ezeket a stringeket?
  - pl. karakterenként valamilyen lezáró jellel
  - pl. valamilyen szűrőket használva, amik támogatják a sorok küldését/fogadását
- A szerver milyen formában válaszoljon?
- A szerver válasza után bontsák a kapcsolatot, vagy egyéb műveletek következhetnek?

Esettanulmány – egy lehetséges megoldás

A fenti feladatot megvalósító protokoll

Kliens oldalon	Szerver oldalon
1. A kapcsolat felépítése	1. Várakozás a következő kliensre
2. Stringek darabszámának átküldése egész számként az output streamen	2. A stringek darabszámát tartalmazó egész szám beolvasása (N) az input streamről
3. A stringek átküldése egyenként az output streamen	3. N darab string beolvasása az input streamről
4. Az eredmény kiolvasása az input streamről	4. A leghosszabb string átküldése az output streamen
5. A kapcsolat lezárása	5. A kapcsolat lezárása goto 1

Implementáljuk ennek megfelelően a kliens és a szerver alkalmazást!

Házi feladat

A,

Készítsen fordítási szolgáltatást végző szervert. Jellemzői:

- A szerver rendelkezzen néhány szavas „adatbázissal” az induláshoz
- A szerver folyamatosan működjön és egy porton várja a kliensek kapcsolódását
- Belépéskor a kliens választhasson az alábbi funkciók közül
  - Új szópár felvitele: a két megadott szót vegye fel a szerver az adatbázisba
  - Fordítás: a megadott szót keresse ki az adatbázisból és adja vissza a másik nyelven
  - Szótár: a szerver adja vissza az általa ismert összes szót
  - Kilépés

B,

Készítsen kliens-szerver alapokon off-line üzenetküldő alkalmazást. Jellemzői:

- A szerver alkalmazás leállítás nélkül működik és egy porton várja a klienseket
- Egy kliens csatlakozhat a szerverhez, megadva a felhasználó nevét és jelszavát. Első belépéskor a felhasználó bármilyen jelszót megadhat, ezt követően azonban már csak ezzel léphet be
- Csatlakozást követően lekérdezheti a rendszerben levő felhasználók neveit
- Ugyanitt küldhet üzenetet bármelyik felhasználónak (küldő, címzett, üzenet)
- Ha egy már ismert kliens lép be a szerverre, belépéskor jelenjenek meg az üzenetei

Ajánlott irodalom

Az óra anyagához kapcsolódó irodalom

- Nyékyné Gaizler Judit: Java 2 útikalauz programozóknak 1.3 II.; ELTE TTK Hallgatói alapítvány, Budapest
  158 – 193. o.

  408 – 418. o.
- The Java™ Tutorials: Basic I/O
  http://java.sun.com/docs/books/tutorial/essential/io/index.html
- Java Networking Overview

  http://java.sun.com/j2se/1.4/pdf/networking.pdf