Pázmány Péter Katolikus Egyetem
Információs Technológiai és Bionikai Kar

Java hálózatkezelés

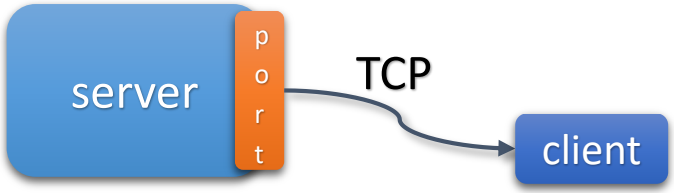
Socket, URL-ek kezelése, Szerver-kliens kapcsolat

Hálózatkezelés alapfogalmai

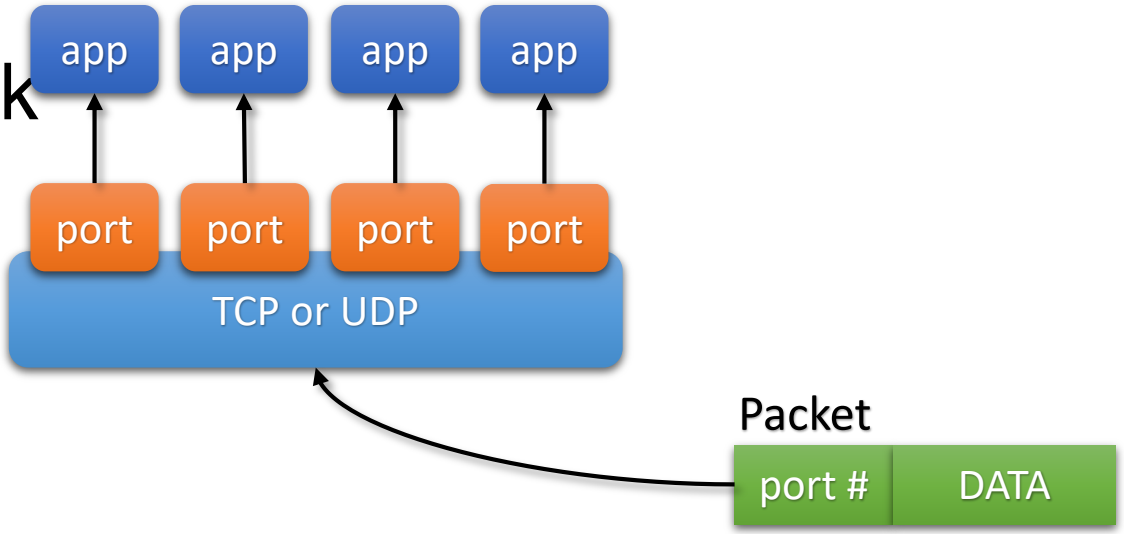
- IP cím: a számítógép (al)hálózaton történő egyedi azonosítására szolgál
- port: a számítógép hálózati interfészén található, számokkal címzett „bejáratok”
- socket: kapcsolat egy port és egy program között
  - ezen keresztül tud egy program írni/olvasni egy portra/ról
  - http://docs.oracle.com/javase/tutorial/networking/overview/networking.html
- Számítógép hálózatok tárgy:
  - http://users.itk.ppke.hu/~mpasztor/netora/2015-osz/

Hálózatkezelés alapjai

- A hálózaton lévő két számítógép (leggyakrabban) TCP protokollon keresztül kapcsolódik egymáshoz:

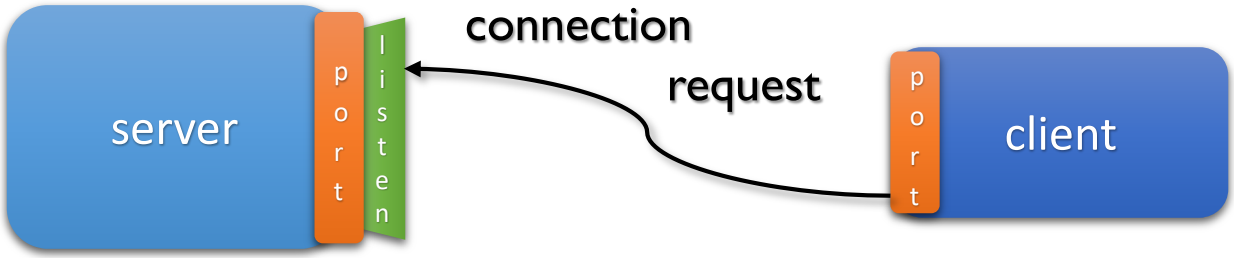


- Az operációs rendszer a hálózat felé portokat hoz létre:
- a portokhoz programok kapcsolódnak, amelyek fogadják az érkező adatokat

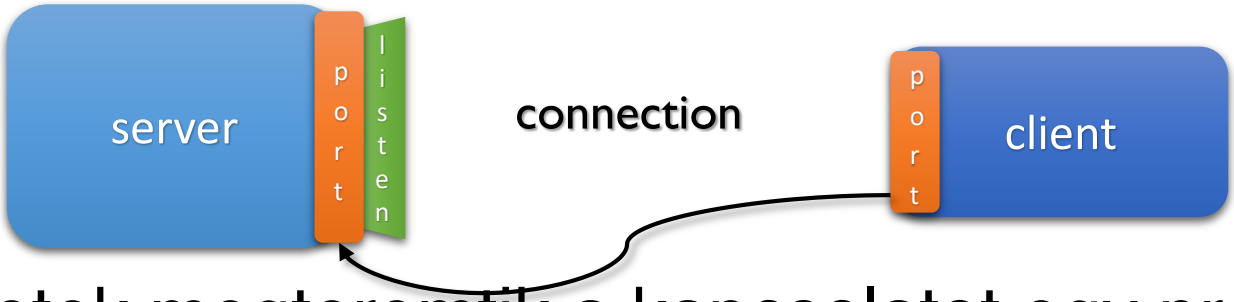


Hálózatkezelés alapjai

- A szerver a megadott porton várja a csatlakozási kérelmeket.



- A szerver fogadja a kérelmet és kezeli a kapcsolatot, mindemellett tovább tud figyelni a porton.



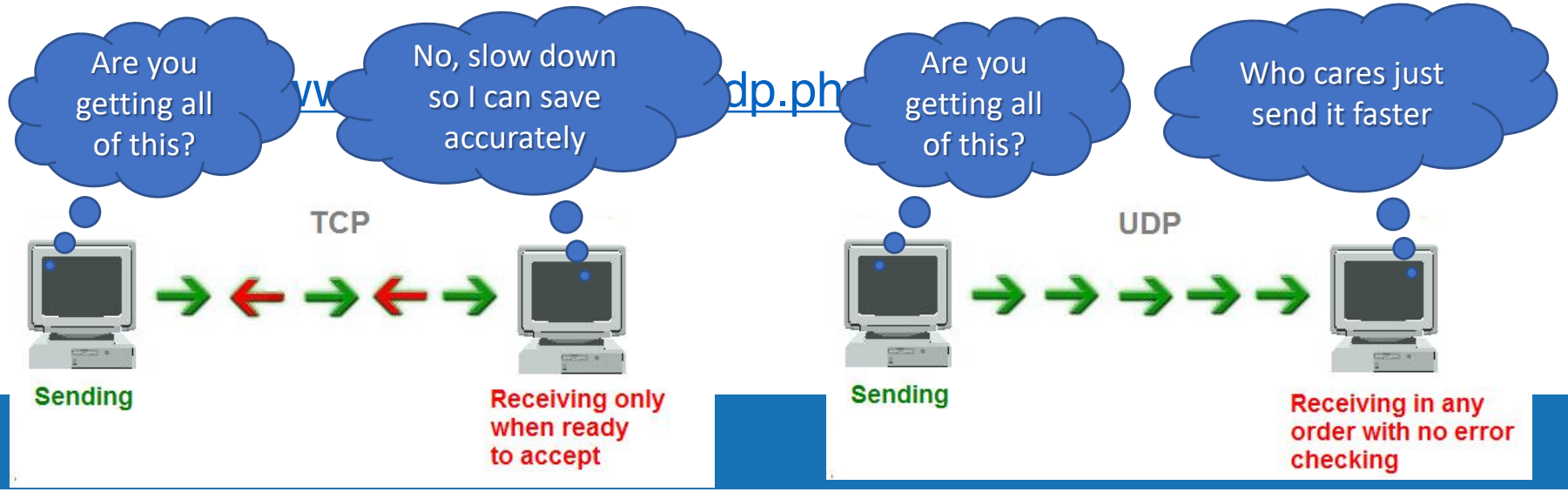
- A socketek megteremtik a kapcsolatot egy program és az adott port között.

TCP/IP Ethernet hálózaton

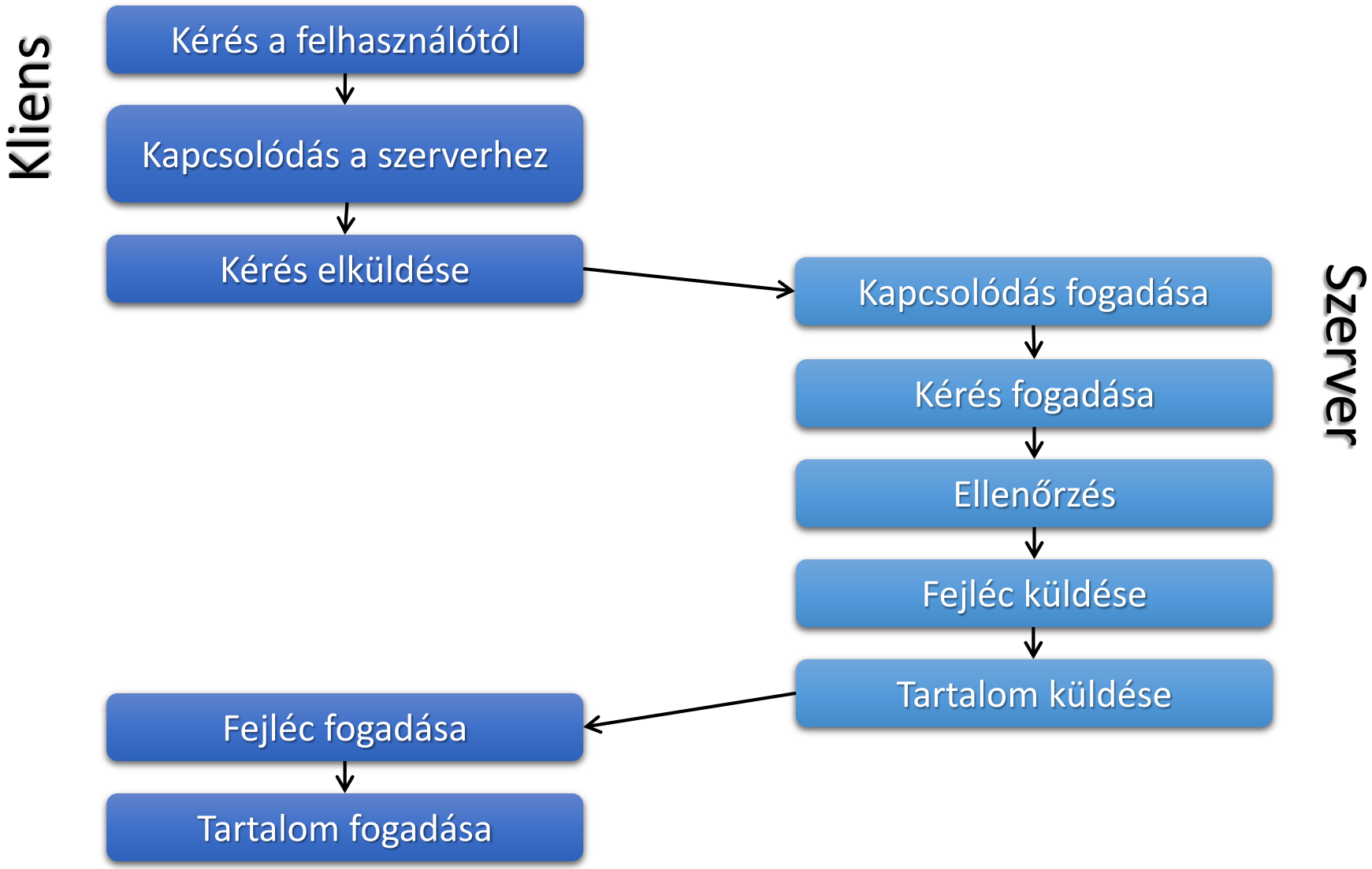
- Az interneten használt kommunikációs formák egy 7 rétegű rendszert alkotnak (ISO/OSI layers)
- Ennek „Transport Layer” rétege felelős az adatok továbbításra történő felkészítéséért
- A TCP (Transmission Control Protocol) illetve az UDP (User Datagram Protocol) egy-egy megvalósítása a Transport Layer funkcionalitásának
- A hálózaton Ethernet csomagok utaznak, amelyek IP csomagokat tartalmaznak, amiken belül TCP csomagokat találunk.
- Minden csomagnak gondosan kialakított fejléce van, ami sok-sok adatot tartalmaz a csomag tartalmára, céljára, stb. nézve.
- Érdeklődőknek: http://hu.wikipedia.org/wiki/OSI_modell

TCP vs. UDP

- TCP: kontrollált adatfolyam, garantált csomagsorrend, ACK üzenetek (acknowledgement – visszaigazolás), tehát lassabb, de biztosabb transzport. Ez ellenőrzött kommunikációt biztosít.
- UDP: visszaigazolás nélküli, gyors, de ellenőrizetlen sorrendiségű és nincs protkoll-szintű csomagvesztés-ellenőrzés (ezt a felsőbb rétegnek kell – igény szerint – megvalósítani). Ez adatfolyam-központú kommunikáció (p2p, skype, …).

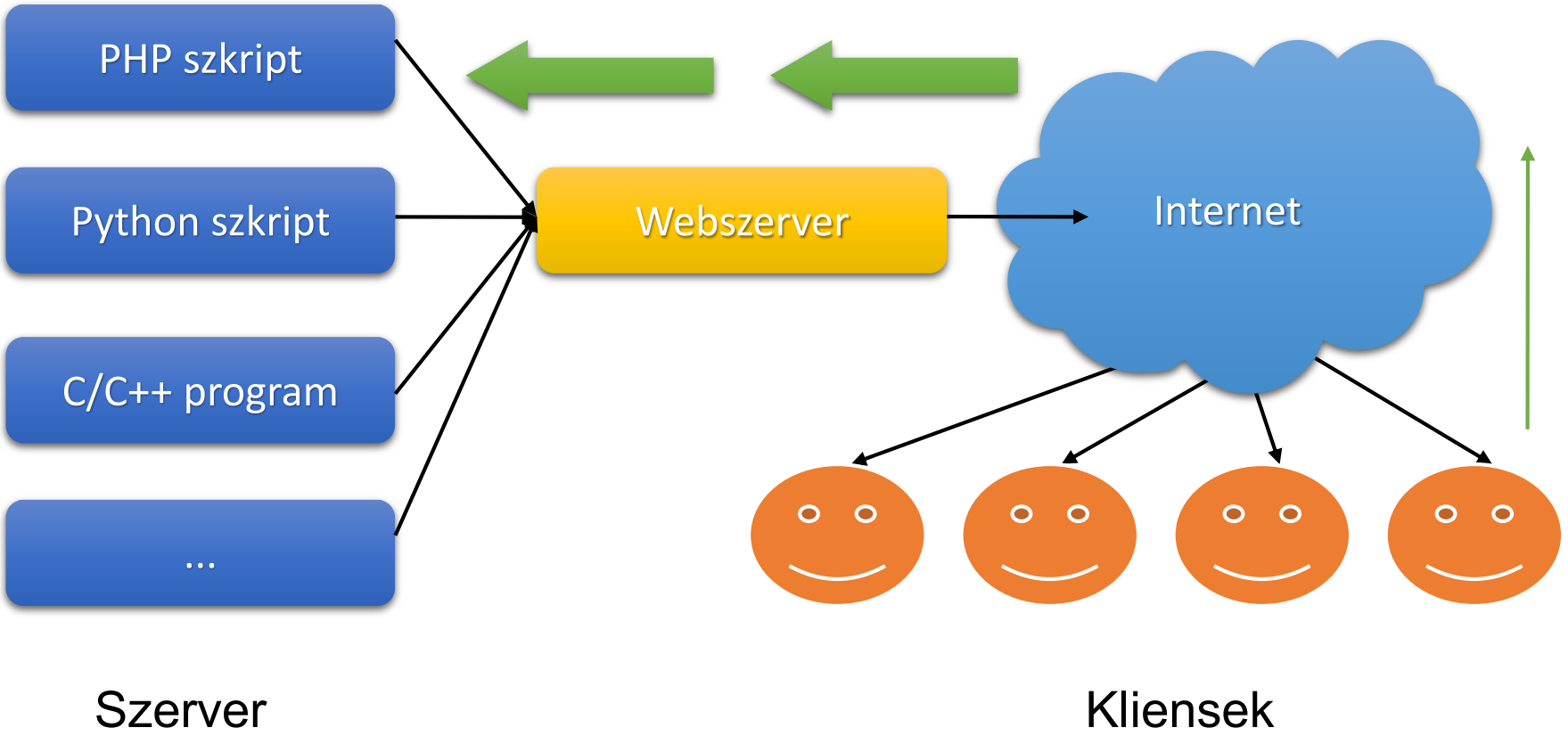


Internetes kommunikáció



Webszerverek

- Olyan szerveroldali programok, amelyek a rajtuk keresztül közzétett tartalmat internetes kapcsolaton keresztül publikálják



És mi van a webszerver mögött?

- Ahogy láttuk, a webszerverek csak „közvetítők”: a szerveren található tartalmat a kliens felé továbbítják. Ezen kívül kiszolgálják a kliensek lekéréseit (válaszolnak rájuk).
- A webszerver mögött futó programok klasszikusan 3 nagy csoportra oszthatók:
  - Adatrendező programok („üzleti logika”)
  - Adatbázisok, adattárolók
  - Megjelenést elrendező programok (template-rendszerek)
- Sokféle szerver-oldalra írt program létezik, rengeteg szolgáltatás ellátására, sok-sok (cél)nyelven, stb.
- Kommunikáció vizsgálatához programok: Wireshark, Netcat.
- Az itt bemutatottnál sokkal összetettebb a kép, de a tárgy keretében nincs időnk ennél mélyebben tárgyalni a dolgokat.

Webszerverek

- HTTP szabvány használata (http://www.w3.org)
- HyperText Transfer Protocol
- Webszerverek és klienseik közötti kommunikációra
- URL-eket fogadnak és dolgoznak fel, ennek alapján válaszolnak
- Kimenetük egy szöveges folyam, a hálózat felé továbbítva

Pár szó a HTTP szabványról

- Üzenetformátum:
  - <start-line>
    <message-headers>
    <empty-line>
    [<message-body>]
    [<message-trailers>]
- Válaszkódok a kliensnek:
  - 1xx Informational
    2xx Success
    3xx Redirection
    4xx Client Error
    5xx Server Error
- továbbiak (csak ínyenceknek!): http://upload.wikimedia.org/wikipedia/commons/6/65/Http-headers-status.gif
- További részletek: http://www.tcpipguide.com/free/t_HTTPGenericMessageFormat.htm

HyperText röviden

- HTTP: eredetileg HTML állományok továbbítására
- HTML: HyperText Meta Language
  - Az utasítások ún. tag-ek, amelyek <> között vannak. Legtöbbjüket le is kell zárni, ezt a </tagname> utasítással tesszük.
- Egy HTML dokumentum szerkezete (nagyon vázlatosan):

```
<html>
    <head>
            <title>Fejléc szövege</title>
    </head>
    <body>
            Tartalom
    </body>
</html>
```

HyperText röviden

- A <body></body> tag-ek közé helyezhetjük a tartalom-formázó utasításokat, pl.:
  - Kép beszúrása: <img src=”image.jpg”>
  - Hivatkozás:
    Hivatkozásszövege
  - Felsorolás:

    listaelem 1.
    listaelem 2.

  - Szövegformázás: vastagon szedett, dőlt, sortörés:
  - Igazítások: középre igazított

URL osztály

- URL = Uniform Resource Locator
  - http://zzz.aa/vvv/, sftp://valaki:jelszo@itt.hu:32 prtkl://uname:pwd@hostname:port/path?param#anchor https://www.gmail.com
  - az Interneten megtalálható helyek szabványos elérhetősége
- Java beépített URL-kezelő osztállyal rendelkezik:
  - `URL webcim = new URL(spec);`
  - Ez az osztály mindenféle hasznos információval tud szolgálni az adott URL-t illetően (host, port, stb.)
- Kiváltott kivétel: MalformedURLException

URLConnection

- Az `URL` által leírt kapcsolatot kezeli
- Kiváltott kivétel: `IOException`
- Megfelelő tagfüggvényei segítségével visszakapjuk az általa kezelt stream-eket:
  - `URL testUrl = new URL("http://www.google.com");`
  - `URLConnection testConn = testUrl.openConnection();`
  - `InputStream testStream = testConn.getInputStream();`
- Ezt a stream-et ezután természetesen már kezelhetjük a megszokott `BufferedReader`-ünkön, vagy más IO osztályon keresztül.
- Az `URLConnection` alatt felépült kapcsolat egy olvasás/írás után automatikusan lezárul. (http://en.wikipedia.org/wiki/Stateless_protocol)

URLConnection

- A létrehozott példány paraméterein és metódusain keresztül küldhetünk üzeneteket a szervernek a HTTP fejléc módosításával, pl.: sütik kezelése (a szerver által a kliensre elhelyezett kisméretű információs fájlok)
  - `URL url = new URL("http://java.com/");`
  - `URLConnection conn = url.openConnection();`
  - `conn.setRequestProperty("Cookie", "foo=bar");`
- a szerver által a kliens azonosítására használt adatok módosítása (lásd később)

Socket

- Vannak olyan esetek, amikor nem használhatjuk az URL osztályt a kommunikáció során
- A `java.net.Socket` teszi lehetővé, hogy nyers bájtcsomagokat küldjünk egy szerver felé
- Megadott portra csatlakozik
- Alacsonyabb szintű, mint az `URLConnection`

Socket - adatküldés

```
Socket sock = new Socket("www.google.hu", 80);
InputStream is = sock.getInputStream();
OutputStream os = sock.getOutputStream();
.
.
.
sock.close(); // a streamek lezárása
```

HTTP metódusok

- HTTP protokoll által meghatározott szerver-kliens kommunikációs formák leggyakoribb: POST, GET (ld.: http://en.wikipedia.org/wiki/HTTP)
  - POST: a szervernek küldött HTTP fejlécbe írja az átküldendő adatot
  - GET: az URL-ben továbbítja az adatot, pl: index.php?x=5&y=asdf

Konklúzió

- A webszerver és a kliens közötti kommunikációt jól definiált protokollok írják le.
- Java programunk bármikor átveheti egy böngésző helyét (Java API segítségével is tudunk böngészőt írni)
  A HTTP protokoll segítségével üzeneteket tudunk küldeni a webszervernek → interaktív kapcsolat alakítható ki

Feladatok

- URL beolvasása
  - `URLReader` – tulajdonságok
    - próbáljuk ki kedvenc webcímeinkre!
  - `URLConnection` kezelése: olvasás, írás:
    - Megadott URL-en lévő tartalom beolvasása
    - POST illetve GET metódussal adatküldés egy távoli programnak, amely azt átveszi, és válaszol.
- Google Fight
  - A Google által az adott keresőkifejezésekre adott találatok számának összehasonlítása.
  - Kicsit trükközni kell, mert a Google nem szereti, ha szkriptből használjuk a keresőmotorját.
  - Feladat: a kommenteket követve egészítsd ki a kódot!

Szerver-kliens kapcsolat

Socket

- Hálózati kommunikáció alapja
- Kapcsolatkiépítés:
  - `bind()` – ezen a címen és porton keresztül
  - `connect()` – erre a címre és portra csatlakozunk
  - (Megj.: ha nem hívunk külön `bind()`-ot, a `connect()` esetén az automatikusan lefut)
    (Megj.: ezek a paraméterek Java esetén a konstruktorban is megadhatók)
  - `getInputStream()` - ha fogadni akarjuk a bejövő adatokat, akkor azt egy `InputStream`-en keresztül tudjuk
  - `getOutputStream()` - ha küldeni is szeretnénk valamit a másik félnek
- Mit tegyünk, ha azt akarjuk, hogy mások tudjanak hozzánk csatlakozni?

Szerver-kliens kapcsolat

- `ServerSocket` osztály fogadja a kapcsolódásokat Fontosabb műveletei:
  - `bind()` - milyen címen és porton várja a kapcsolatokat
  - `accept()` - egy kliens fogadása, blokkol, amíg nincs kapcsolódó számítógép
- De a tényleges kommunikáció már a `Socket`-en keresztül történik, amelyet az `accept()` visszaad.

Egy kliens fogadása

```
ServerSocket server = new ServerSocket(55555);
Socket client = server.accept();
...
client.close();
server.close();
```

- Nézzük meg a `ping` csomagot!

Néhány hasznos gondolat

- A szerver-kliens kommunikáció során 10000 fölötti portszámot válasszunk (közmegegyezés szerint ezek a szabadon használható portok)
- Ne felejtsünk el minden használt socket-et és stream-et lezárni kilépés előtt!
- Socket-re írás esetén használjuk a `PrintWriter` két paraméterű konstruktorát (`autoFlush = true`, azaz a `println()` automatán üríti a puffert)
- Ez operációs rendszer függő, és ha ettől független kommunikációt szeretnénk, akkor simán `print()` -et kell használnunk, és manuálisan meghívni utána a `flush()` -t

Néhány hasznos gondolat

- Több program futtatása párhuzamosan Eclipse-ben, több konzol használata
- Figyeljünk, hogy hány példányát futtatjuk a programnak (nem fut-e még egy korábbi)!
- Figyeljünk, hogy éppen melyik program konzolján vagyunk!
- Futtathatjuk parancssorból:
  Start > Futtatás: cmd > java zzz.class -cp classpath

Szálasított kapcsolat

- Eddig csak egy klienssel beszélgettünk, ami már önmagában is elég izgalmas, de a nagyvilágban ritkán használható. Általában egyszerre (elvileg) akármennyi klienst ki akarunk szolgálni.
- Kliens fogadása mindig az `accept()` metódusnál történik. Tehát „csökkentsük” két `accept()` közti időt.
- Alapötlet: a szerver minden egyes klienssel történő kommunikációját külön szál kezeli, míg a main szál továbbra is a porton figyel (új bejövő kapcsolatokra várva).

Szálasított kapcsolat

- Kukkantsunk bele az `epicjoketeller` csomagba!
- Egyszerre sokan kapcsolódtok egy géphez, a szerver mindenkit képes kiszolgálni.

Szálasított kapcsolat

- Konklúzió:
  - A szerver külön szálat indít minden bejövő kapcsolatra
  - A kliensek semmit sem tudnak egymás létezéséről, csak a szerver fele tekintő input-output streamekkel foglalkoznak.
- A szerver-kliens modell eléggé hasonló általában. Láthattuk, hogy az egy kliens kezelése-több kliens kezelése csupán a szálak indításában különbözött, innentől kezdve pedig a program váza minden esetben ugyanaz lesz. Ezért ezt egyszer kell megérteni.
- A való életben azért vannak más technikák is több kliens kezelésére (pl. select, poll, epoll).

Szerializáció

- Alapgondolat: a Java objektumok (szerkezetükkel és tartalmukkal együtt) „mozgathatóak” legyenek.
  - Két hálózati egyed között
  - A program két futása között (kimenthető legyen)
- Praktikus megközelítés: ne kelljen mindegyik objektumhoz fáljba/stream-ba dump-oló, és onnan kiolvasó függvényeket írni (`toString()` metódus).
- Ötlet: írjuk ki a stream-re a Java objektum bájtfolyammá alakított képét.

Szerializáció

- A szerializáció megvalósítása az `InputStream` ill. `OutputStream` leszármazottaiban van:
  - `ObjectInputStream` / `ObjectOutputStream`
  - Ezekben található `readObject`, `writeObject` metódusok
  - Szükséges: a tárolandó objektum implementálja a `Serializable` interfészt
- Vigyázzunk: ha módosítunk az osztály szerkezetén, vagy nevén, akkor a korábbi verzió szerializált példányát nem biztos, hogy vissza fogjuk tudni tölteni!
  - lásd: `serialVersionUID`

    https://docs.oracle.com/javase/6/docs/platform/serialization/spec/class.html

Szerializáció

- Ha egy objektumban valamely mező szerializálása nem szükséges, kikapcsolhatjuk a mező definíciójába írt `transient` kulcsszóval
- Mi magunk is megírhatjuk/vezérelhetjük a szerializációt, ehhez az alábbi függvényeket kell deklarálni:
-
```
private void readObject(java.io.ObjectInputStream stream)
throws IOException, ClassNotFoundException
```
-
```
private void writeUnshared(java.io.ObjectOutputStream stream)
throws IOException
```

Szerializáció

- További tudás és példák: https://docs.oracle.com/javase/tutorial/jndi/objects/serial.html
- https://docs.oracle.com/javase/8/docs/technotes/guides/serialization/examples/index.html
- Írjunk példakódot:
  - `HashMap` feltöltése mindenféle adattal
  - Kiírás az alapértelmezett `writeObject` metódussal
  - Visszaolvasás `readObject`-tel
  - Látható, hogy akár hálózati stream-be is írhattuk volna, nem kötelező fájlba írni
- Ha egy objektumot kétszer írsz ki a streamre, és közben megváltozott, akkor az eredetit fogja kiírni!
  - Ennek elkerülésére a második kiírás előtt hívd meg a stream `reset()` metódusát
  - Vagy használd a `writeUnshared` – `readUnshared` párt

https://docs.oracle.com/javase/8/docs/api/java/io/ObjectOutputStream.html#writeUnshared-java.lang.Object-

Gyakorló feladat G08F01

- Írj programot, amely a következőt teszi:
  - grafikus felületet ad, egy szövegterülettel, egy szövegmezővel és egy gombbal
  - beolvassa a szövegmegzőbe beírt keresőszót, és a következő URL-t formázza belőle: http://shop.ebay.com/i.html?_nkw=str (ahol str a szövegmezőben talált string, melyben a space-eket cseréld + jelre!)
  - írd ki, hogy hány találat van összesen a keresőszóra
  - írd ki a talált elemek neveit! (tipp: nézelődj a forrásban a class=”vip” string környékén). Természetesen elég csak az első visszaadott oldal tartalmát vizsgálni!
  - A nevek mellé írd ki a találatok árát is!
  - írd ki az első oldalon látható találatok átlag-árát.
  - használj reguláris kifejezéseket a keresésre!

Gyakorló feladat G08F02

- Írj egy egyszerű linkkövető programot:
  - A program induláskor beolvassa a konzolról egy http:// címet.
  - Majd megkeres minden   taget.
  - A linken levő oldalt is látogassa meg, és járjon el ugyanúgy.
  - Az egyes meglátogatott weboldalakat adatait egy HashMap<String, Integer> tárolóban tartsuk nyilván
  - Ha egy oldalt legalább 2 alkalommal meglátogatott, akkor azt újra ne töltse le.
    - (Ezt az értéket konstansként tárold el, hogy könnyen meg lehessen változtatni.)
  - A bejárás végén listázzuk ki a konzolra a statisztikát, látogatás szerint csökkenő sorrendben.

Gyakorló feladat G08F03

- Írj programot, amely a következőt teszi:
  - Grafikus felületet ad, egy szövegterülettel, két szövegmezővel és egy gombbal.
  - Az egyik szövegmezőbe egy webcímet kell megadni, a másikban pedig egy keresőszót.
  - A gomb megnyomásakor ellenőrizd, hogy a megfelelő helyre megfelelő szöveg került (jó-e a webcím formázása)
  - Ha nem jó, kezeld valamilyen módon. (hibajelzés, átformázás, pl. http:// automatikus kiegészítése)
  - Írd ki, hogy hány találat van a megadott oldalon a keresőszóra.
  - Ha a találatok között van link, írd ki a linket a szövegterületre.
  - Ha a találatok között van kép, jelenítsd meg.
  - Használj reguláris kifejezéseket a keresésre!

Gyakorló feladat G08F04

- Hozz létre egy grafikus felületet, egy szövegdobozzal, két gombbal és két szövegterülettel.
- A szövegdobozba egy URL linket kell írni. Az egyik gombra kattintva a program csatlakozik a megadott linkhez. Az egyik szövegterületen megjelennek az URL adatai (protocol, authority, host, port, path, query, filename, ref). Hibás URL esetén ugyanezen a szövegterületen jelenítsük meg a hibát. A másik szövegterületen megjelenik a teljes HTML fájl.
- A másik gombra kattintva felugrik egy TextInputDialog (segítség itt: http://code.makery.ch/blog/javafx-dialogs-official/), ahol meg lehet adni egy keresőszót. Ezt követően a második szövegterületről eltűnik az eredeti HTML fájl, és csak azok a sorok jelennek meg, amelyekben megtalálható a keresett kifejezés.

Gyakorló feladat G08F05

- Írjatok hálózaton játszható csigaversenyt!
  - Legyen egy szerver, ez várja a bejövő kapcsolatokat, ők lesznek a versenyzők.
  - A szerver rendelkezzen konzolos interfésszel, ahol a START parancs begépelése után az addig csatlakozott játékosokkal elindul a verseny.
  - A játékosok úgy tudnak előrelépni, ha a szervernek elküldik a GOGOGO parancsot. Minél gyorsabban teszik, annál gyorsabban ér célba a csigájuk.
  - A játéknak akkor van vége, amikor mindenki célba ért.

Gyakorló feladat G08F06

- Írjatok olyan játékot, ahol egymást kell felrobbantani!
  - A szerver várja a bejövő kapcsolatokat, ők lesznek a játékosok.
  - A szerver rendelkezzen konzolos interfésszel, itt a START parancs begépelésével lehet a játékot indítani.
  - Induláskor a szerver indít egy szálat, ez lesz a bomba, ami egy konstruktorban megadható int értéket csökkent másodpercenként eggyel. A játék indulásakor az érték legyen random.
  - A szerver ezek után "odaadja" valakinek a bombát, ezt jelzi is neki (pl. elküldi neki a "You have the bomb!" karakterláncot) (közben persze a bomba tovább számol). A játékos a bombát a TOVABBADOMABOMBAT parancs elküldésével adhatja vissza, ilyenkor ismét egy random emberhez kerül (tehát lehet, hogy visszakerül az illetőhöz).
  - A játék addig tart, amíg a bomba fel nem robban. Ezt a szerver minden műveletnél ellenőrizze. Az a játékos veszít, akinél a bomba volt, amikor felrobbant.

Gyakorló feladat G08F07

- Akasztófa
  - A házi feladatban a népszerű akasztófa szókitaláló játék szerverrel játszható, Java verzióját kell elkészítenetek.
  - A kliensek GUI-val rendelkeznek, a GUI-ban egy szövegmező látszik az éppen az éppen játszott, egyes karaktereiben hiányos szóval, továbbá egy másik szövegmező az új karakter beviteléhez, egy gomb az elküldéshez, ezen kívül egy JProgressBar, ami jelzi, mennyit próbálkozhat még a játékos.
  - A szerver több klienst képes kiszolgálni, a kliens csatlakozásakor elküld neki egy véletlen szót, egyes karaktereit elfedve. A kliens ekkor küld egy karaktert. A szerver ezt feldolgozza, és ha volt ilyen karakter, akkor kiküldi a szót a karaktert felfedve, ha nem volt, elküldi az élet csökkenését jelző üzenetet. Ha az élet nullára csökken, vagy a teljes szó kitalálásra került, a játéknak vége.
  - Az implementáláskor figyeljetek oda, hogy a GUI ne fagyjon ki, a szerver szálasítot kapcsolatot használjon a több kliens kezelésére, tartsátok be az OOP alapelveket, ügyeljetek a láthatóságokra, kezeljétek helyesen a kivételeket és tartsátok be a névkonvenciókat.

Gyakorló feladat G08F08

- A feladat egy hálózaton játszható aknakereső játék megírása (konzolra).
- A pálya 20x20-as 40 random elhelyezett bombával, amit a szerver generál.
- A játékot legalább 2, legfeljebb 4 kliens játékos játszhatja egyszerre.
- A játék körökre osztott, egy körben mindenki „lép” egyet. A lövés helyét sor-oszlop koordinátával lehet megadni az alábbi formában: „2 – 3”, ahol 2 a 2. sor számát, míg a 3 a 3. oszlop számát jelöli. A kliens elküldi ezt az üzenetet a szerver felé, amit a szerver feldolgoz, majd az eredményt elküldi MINDEN kliens számára.
- Ha nincs találat, akkor azon a helyen egy szám jelzi, hogy hány bomba található a közvetlen környezetében. (nem szabad továbbfejteni a 0-s mezőt sem) Amennyiben találat van, ott felrobbant egy bomba, egy számláló növelésével jelzi, hogy az adott felhasználó hány bombát talált meg addig. Minden körnél jelenjen meg minden felhasználó számára a pálya, és az is, kinek hány bombája van.
- A játékot az nyeri, aki az összes bomba felrobbantása után a „legtöbb megtalált bomba” címmel rendelkezik.

Gyakorló feladat G08F08 II.

- A program mindennemű hibakezelése tökéletesen működjön, ne fagyjon le az alkalmazás (kliens kilépése, netkapcsolat megszakadása esetén sem).
- Egy játék véget érésekor legyen lehetőség új játék játszására a meglévő játékosokkal. (kérdés feltétele mindenkinél, és ha min. 2 játékos igennel szavaz, az igennel szavazókkal új játék kezdődik)
- A szerver telítettsége esetén (4 kapcsolódott kliens), írja ki az alkalmazás a próbálkozóknak, hogy sajnos a játék elérte a maximális létszámot, próbálkozzon később. Amennyiben 1 kliens kilép, vagy kevesebb, mint 4 játékos játszik, (új játék kezdésénél) legyen lehetőség becsatlakozni a helyébe.
- A szerver külön induljon el, és ahhoz tudjon kapcsolódni a kliens. Tehát ne valamelyik játékos legyen a szerver, hanem külön szerver legyen!

Gyakorló feladat - Chatroulette

G08F09

- Készítsük el a Chatroulette egyszerűsített, konzolos változatát!
- Kliens
  - Csatlakozik a szerverhez
  - Vár amíg nem kap partnert
  - Miután kapott, küldeni tud neki üzenetet, illetve megkapja a partner üzeneteit
- Szerver
  - Fogadja a kliensek kapcsolódását, de amíg nincs legalább 5, mindig egy várakozó halmazba helyezi őket
  - Ha az 5. is csatlakozott, akkor az előző négyből sorsol egyet neki partnernek, és ezután a két kliens egymással beszélgethet, kikerülnek a várakozó halmazból
  - Ezt folytatja tovább a végtelenségig
- Extra pontért, ha az alap megoldás már tökéletes:
  - A szerver 5 perc után dobja ki a klienst, és helyezze vissza a várakozási sorba

Gyakorló feladat G08F10

- Készíts egy egyszerű rajzoló programot
  - Mondjuk, ha le van nyomva az egér gombja, akkor a kurzor nyomot hagy az ablakban
  - Vagy gombnyomással lehet új pontokat felvenni egy töröttvonalhoz
  - Vagy bármi egyéb: a lényeg, hogy egérrel valahogy tudjak rajzolni
- Ezt nevezzük el szervernek, és figyeljen egy meghatározott porton
- Készíts egy kliens programot, amely
  - induláskor megkérdezi a felhasználót, hogy a szerver hol található
  - ezzel az információval felvértezve csatlakozik a szerverre, és program bezárásáig csatlakozva marad
  - a szerver csatlakozáskor elküldi a rajzolmány aktuális állapotát
  - ezt a kliens rögtön megjeleníti
  - a szerver minden módosítást rögtön küldjön ki a csatlakozott klienseknek
  - ezeket a kliens rögtön megjeleníti
- GUI ne fagyjon és csak EDT szálról történjen rajzolás!

Gyakorló feladat G08F11

- Írj egy hálózatban játszható tic-tac-toe játékot
  - A játék körökre osztott, hol az egyik, hol a másik játékos jön
  - Legyen egy szerver aki a játékot kezeli
  - A szerverhez kapcsolódhat két kliens, melyeken egy GUI fut, ami a játék állapotát mutatja
  - A játék akkor kezdődik ha két kliens csatlakozott
  - Ha harmadik kliens is próbál csatlakozni, értesítsük hogy a játék tele van
  - A kliens csak akkor ”léphet” ha az ő köre jön, ekkor a 3x3-mas pálya egy még üres mezőjére kattintva, oda helyezheti a jelét (X vagy O)
  - Kezeljük le azt is ha az egyik kliens kiesik vagy kilép – lehessen a helyére lépni
  - A szerver értesítse a klienseket ha az egyik játékos nyert, azok pedig jelezzék ezt egy üzenettel.