Programozási technológia 2.

Hálózatkezelés

Dr. Szendrei Rudolf

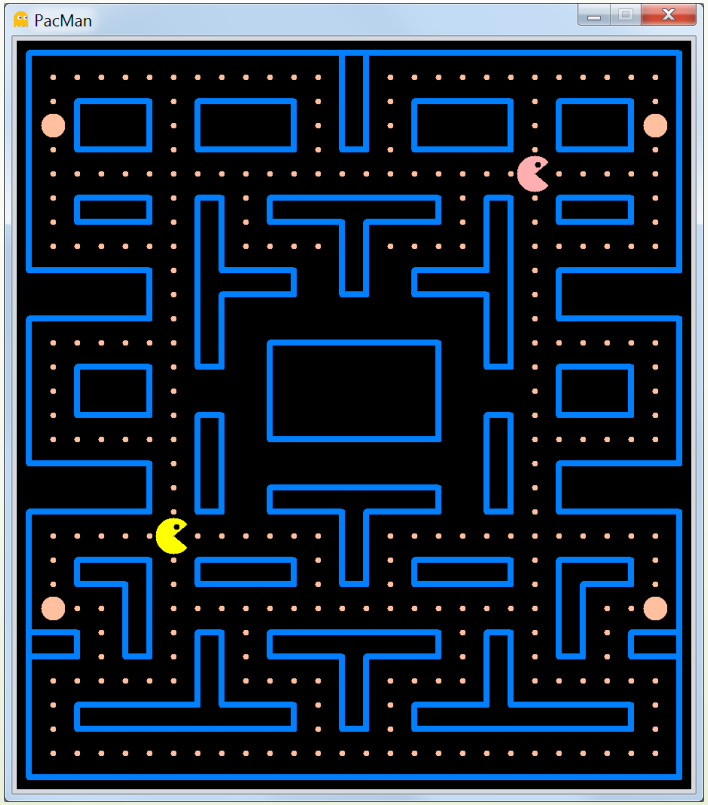
ELTE Informatikai Kar

2018.

Hálózatkezelés

PacMan

- Készítsünk egy szellem mentes két személyes PacMan játékot
- Ha a játékosok PacMan-jei mindent megettek, a játék újraindul
- Alkalmazzunk MV architektúrát



Hálózatkezelés

PacMan – modell

- A játék egy személyes implementációját a korábbi félév alapján már el tudjuk készíteni MV architektúrában
- A két személyes módhoz egy helyett két PacMan-t tárolunk el a modellben.
- A játékosokhoz a W,A,S,D és a ↑,←,↓,→ gombokat rendeljük.
- A modellben két fő osztály jelenik meg
  - Player
    - Tárolja egy PacMan nézési, mozgási, kanyarodási irányát, a pozícióját és a színét
  - GameBoard
    - Tárolja a pálya állapotát, a két PacMan példányt, és megvalósítja a logikát

Hálózatkezelés

PacMan – modell

- GameBoard logika fő részei
  - changePlayerDirection(int player, Direction d){…}
    - Eltárolja a megadott játékos PacMan-jéhez a mozgási irány szándékát a saját PacMan példányban
  - movePlayers() {…}
    - A View időzítője által periodikusan hívott metódus
      - adott egységgel elmozdítja a PacMan-eket, amennyiben nem ütköznek falba
      - mozgatás közben animálja PacMan száját
      - eltávolítja a pályáról a megevett falatokat

Hálózatkezelés

PacMan – több személyes mód

- Alakítsuk úgy át a játékot, hogy a két személy internet hálózaton keresztül tudjon együtt játszani.
- Mivel a hálózatos programokban a hibákat jóval nehezebb felderíteni, ezért a fejlesztést több lépcsőben hajtjuk végre.
  1. Modell állapotonák mentése / visszatöltése
  2. Tesztelés
  3. Modell származtatása „vékony kliens” / szerver szerepekhez
  4. Tesztelés
  5. A modellek hálózatos változatának létrehozása
  6. Tesztelés

Hálózatkezelés

PacMan – vékony kliens architektúra

- A vékony kliens lényege, hogy csak megjelenítőként funkcionál, a felhasználói interakciókat a szerverhez továbbítja.
- A szerver minden információval rendelkezik, megvalósítja a logikát, a klienseknek pedig csak a megjelenítéshez szükséges adatokat küldi át.

Hálózatkezelés

Adatok reprezentációja

- Vezessük be a Player és GameBoard osztályokban az állapot mentésére / visszatöltésére szolgáló metódusokat.
- A Serializable interfész használata túlságosan költséges, pl. a Player osztályt 582 bájttal képes csak reprezentálni.
- A Player osztály kliensben megjelenítendő állapotához 6 bájt is elegendő:
  - 1 Byte - nézés iránya (faceDir),
  - 2x2 Byte - pozíció (smoothPos),
  - 1 Byte - száj nyitottsága

Hálózatkezelés

PacMan – Player osztály – állapot mentés / betöltés

```
public byte[] toByteArray(){

  ByteBuffer bb = ByteBuffer.allocate(representationSize);
  bb.putShort((short)smoothPos.x);
  bb.putShort((short)smoothPos.y);
  bb.put((byte)mouth);
  bb.put((byte)faceDir.ordinal());
  return bb.array(); // length = 6 bytes

}

public void updateFromStream(InputStream is)
            throws IOException {

  DataInputStream dis = new DataInputStream(is);
  int sx = dis.readShort();
  int sy = dis.readShort();
  pos.setLocation(sx / scale, sy / scale);
  scaledPos.setLocation(pos.x * scale, pos.y * scale);
  smoothPos.setLocation(sx, sy);
  this.mouth = dis.readByte();
  this.faceDir = Direction.values()[dis.readByte()];

}
```

Hálózatkezelés

PacMan – GameBoard osztály – állapot mentés

```
public byte[] toByteArray() {

  int numBytes = width * height + players.size() *
                 Player.representationSize;

  ByteBuffer bb = ByteBuffer.allocate(numBytes);
  for (int i = 0; i < players.size(); i++){
    bb.put(players.get(i).toByteArray());
  }

  for (int y = 0; y < height; y++){
    for (int x = 0; x < width; x++){
      // mark the differences
      bb.put((byte)((table[y][x] != origTable[y][x]) ? 1 : 0));
    }
  }

  return bb.array();

}
```

Hálózatkezelés

PacMan – GameBoard osztály – állapot betöltés

```
public final synchronized void fromByteArray(byte[] array){
  if (array == null || array.length != representationSize) return;
  ByteArrayInputStream bais = new ByteArrayInputStream(array);
  try {
    for (int i = 0; i < players.size(); i++){
      players.get(i).updateFromStream(bais);
    }
    int byteIndex = players.size() * Player.representationSize;
    for (int y = 0; y < height; y++){
      for (int x = 0; x < width; x++){
        if (array[byteIndex++] > 0){
          table[y][x] = Item.EMPTY;
        }
      }
    }
  } catch (IOException e){}
}
```

Hálózatkezelés

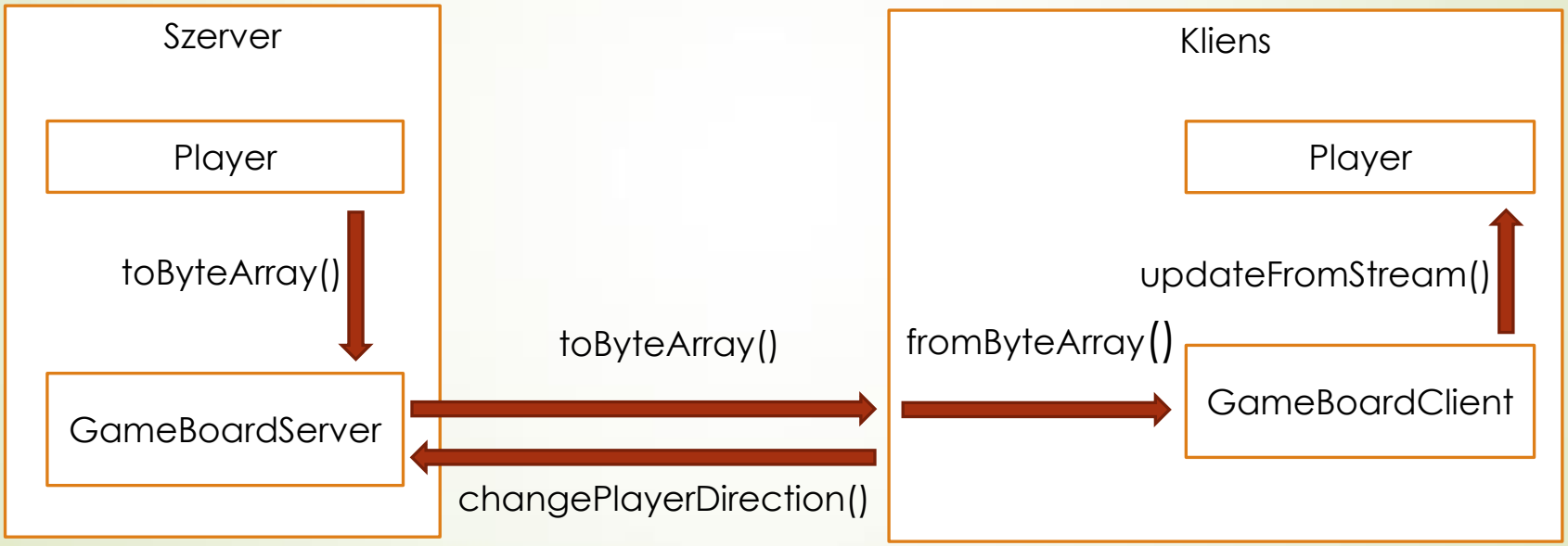
PacMan – Tesztelés

- Az elkészített mentés / visszatöltés metódus párokhoz készítsünk egységteszteket!

Hálózatkezelés

PacMan – Kliens / szerver modellek

- Az elkészített szerializációs metódusokat a kliens/szerver modell a következőképpen fogja alkalmazni



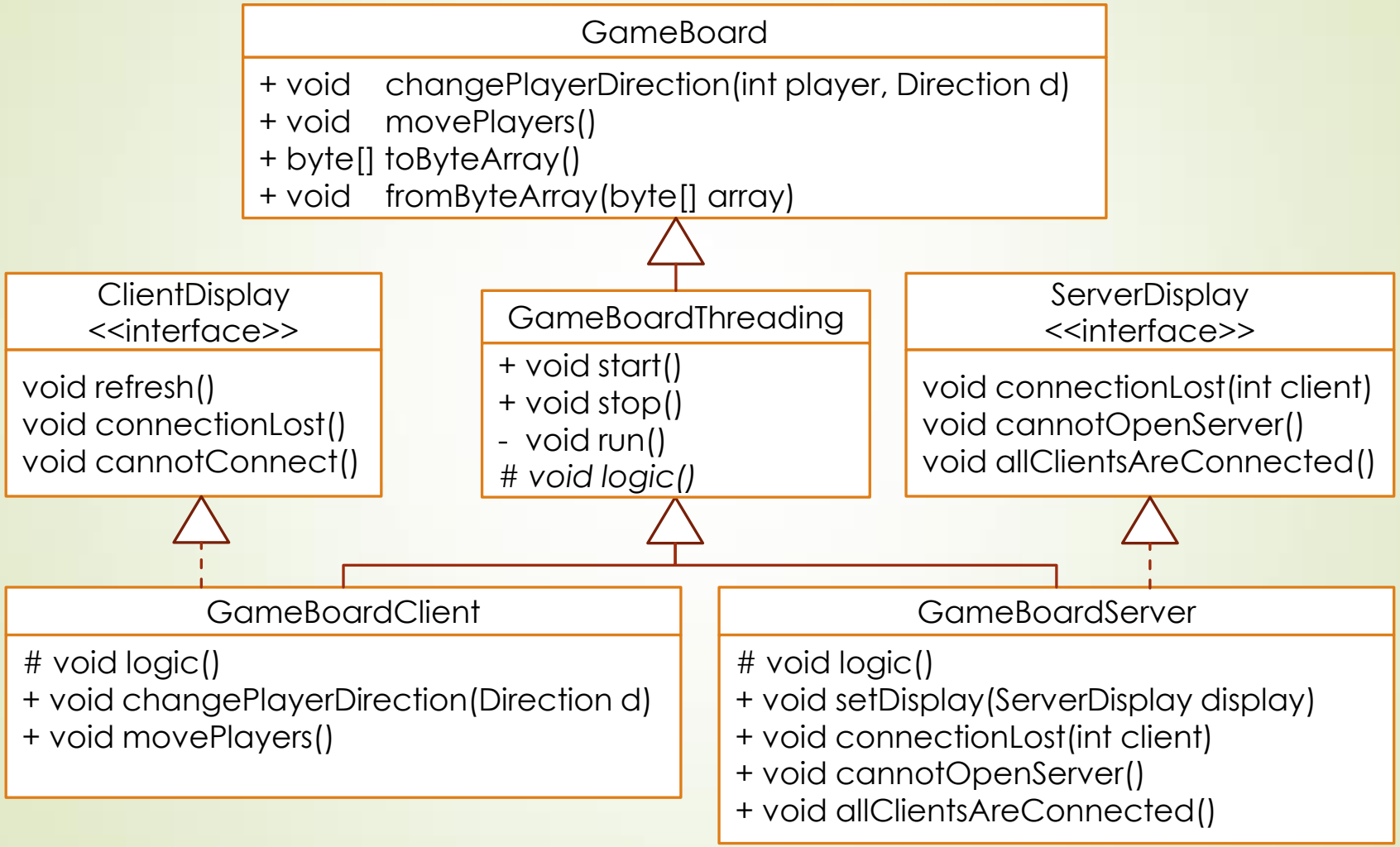
Hálózatkezelés

PacMan – Kliens / szerver modellek

- Mivel a játék logika a szerverben valósul meg, ezért a kliens képernyőjének frissítése nem egy saját időzítőtől, hanem a szervertől kapott jelzésektől fog függni.
- A fentiek miatt elengedhetetlen, hogy a kliens egy külön szálat futtasson a beérkező adatokra való várakozáshoz és feldolgozáshoz.
- A hálózati kommunikáció a szervernél is külön szálat igényel, az ok viszont ebben az esetben a reakció idő minimalizálása. a A kapott utasításokat így nem periodikusan dolgozzuk fel (időzítőre alapulva), hanem egy külön szálon azonnal.
- A modelleket a GameBoard osztályból fogjuk származtatni, azonban a szálkezeléssel kapcsolatos közös részeket egy köztes GameBoardThreading leszármazottba helyezzük.
- A klienseket és a szervert először csak csővezetékkel (Pipe) fogjuk összekötni, hogy teszteljük a módszer helyességét...

Hálózatkezelés

PacMan – Kliens / szerver modellek



Hálózatkezelés

Szálkezelés – emlékeztető

- Védeni kell azon erőforrásokat, melyeket legalább egy szál módosítani tudna úgy, hogy közben egy másik hozzáférhet.
- **Ökölszabályok**
  - A megoldáshoz kell egy obj objektum, aminek a synchronized(obj){…} blokkjában a hozzáférés történik.
    - Ez lehet maga a védendő erőforrás is
  - **Durva megoldás**:
    - synchronized(this){…} a metódusban vagy
    - synchronized metódus, ami állapotot olvas/módosít
  - Az objektum minden írható attribútuma legyen privát.
  - Az objektum típusú attribútumnak a referenciája helyett az objektum másolatát adja vissza a synchronized metódus.

Hálózatkezelés

PacMan – GameBoardThreading

```
public abstract class GameBoardThreading
                extends GameBoard
                implements Runnable{

  private volatile Thread  thread;
  private volatile boolean isRunning = false;
  private final    String  name;

  public GameBoardThreading(String name){ this.name = name; }

  public synchronized void start() { ... }
  public synchronized void stop() { ... }

  @Override
  public void run() { ... }

  protected abstract void logic();

}
```

Hálózatkezelés

PacMan – GameBoardThreading

```
public abstract class GameBoardThreading
                extends GameBoard
                implements Runnable{

  public synchronized void start(){
    if (!isRunning){
      isRunning = true;
      thread = new Thread(this);
      thread.start();
    }
  }

  public synchronized void stop(){
    if (isRunning){
      thread.interrupt();
      while (isRunning){
        try { wait(); }
        catch (InterruptedException ex) {}
      }
    }
  }
}
```

Hálózatkezelés

PacMan – GameBoardThreading

```
public abstract class GameBoardThreading
                extends GameBoard
                implements Runnable{


  @Override
  public void run() {

    logic();

    synchronized (GameBoardThreading.this){
      isRunning = false;
      notifyAll();
    }
    System.out.println(name + ": stopped");
  }

}
```

Hálózatkezelés

Kliens / szerver kapcsolat

- A kapcsolatot a két oldalon InputStream / Outputstream reprezentálja majd, ezért már most figyeljünk a következőkre!
  - Ha nincs elegendő adat a bemeneten, akkor az olvasás blokkolja a szálat!
  - A blokkolt szál nem szakítható meg, és nem érzékeli az olvasás hiányának okát!
- Nem blokkoló megoldáshoz bájt tömböket kell használnunk, melyekbe annyit olvasunk, amennyit éppen lehet.
- Ha rendelkezésre áll a kellő adat, csak akkor dolgozzuk fel.
- A bájt tömböt a ByteArrayInputStream segítségével InputStream-ként dolgozhatjuk fel. Más adattípusokhoz (int, float, double...) létrehozhatunk a szokásos módon DataInputStream-et is belőle.

Hálózatkezelés

Adatkapcsolat megszűnésének érzékelése

- OutputStream
  - Az írási kísérlet IOException-t vált ki.
- InputStream
  - Olvasáskor nem tudhatjuk, hogy a másik fél „elveszett”, vagy csak nincs közlendője.
  - Egy jó megoldás, ha a másik féltől elvárjuk, hogy adott időközönként üzenjen (Ping), így a kapcsolat tétlensége elárulja a kapcsolat megszűntét.
- Az InputStream egyedüli nem blokkoló metódusa az available()
  - Visszatérési értéke jelzi, hogy a fogadó buffere üres-e.
  - A blokkoló read(byte[] buf, int pos, int len) metódussal csak ebben az esetben olvasunk, amennyit éppen lehet.
  - Ha összegyűjtöttük a szükséges adatokat, akkor feldolgozzuk.

Hálózatkezelés

PacMan – GameBoardServer

- Konstruktor
  - Input-/OutputStream tömbben kapja meg a klienseihez tartozó kapcsolatokat
- logic()
  - Ciklusa addig fut, amíg a szál nem kerül megszakított állapotba. A ciklus minden iterációjában
    - Olvas a klienseihez tartozó bemenetekről ha tud, majd ez alapján hívja a changePlayerDirection(...) metódusát
    - Eltávolítja azokat a klienseket, amelyek „eltűntek”
    - Meghívja a movePlayers() metódust a játék léptetéséhez
    - Előállítja a modell bájt reprezentációját a toByteArray() segítségével, és elküldi azt valamennyi kliensének
    - Várakozik a beállított frissítési gyakoriságnak megfelelően

Hálózatkezelés

PacMan – GameBoardClient

- Konstruktor
  - InputStream / OutputStream paramétere a kapcsolatot reprezentálja.
- changePlayerDirection(Direction d)
  - A felhasználói interakciókat kiküldi az OutputStream-re.
- movePlayers() – logika híján üres.
- logic()
  - Ciklusa addig fut, amíg a szál nem kerül megszakított állapotba. A ciklus minden iterációjában vár egy újabb csomagot, amivel frissíti a modellt (ilyenkor a kinézetet is frissíti). Adat hiányában számolja a kapcsolat tétlenségének idejét, és jelzi, ha a kapcsolatot megszakadtnak tekinti.

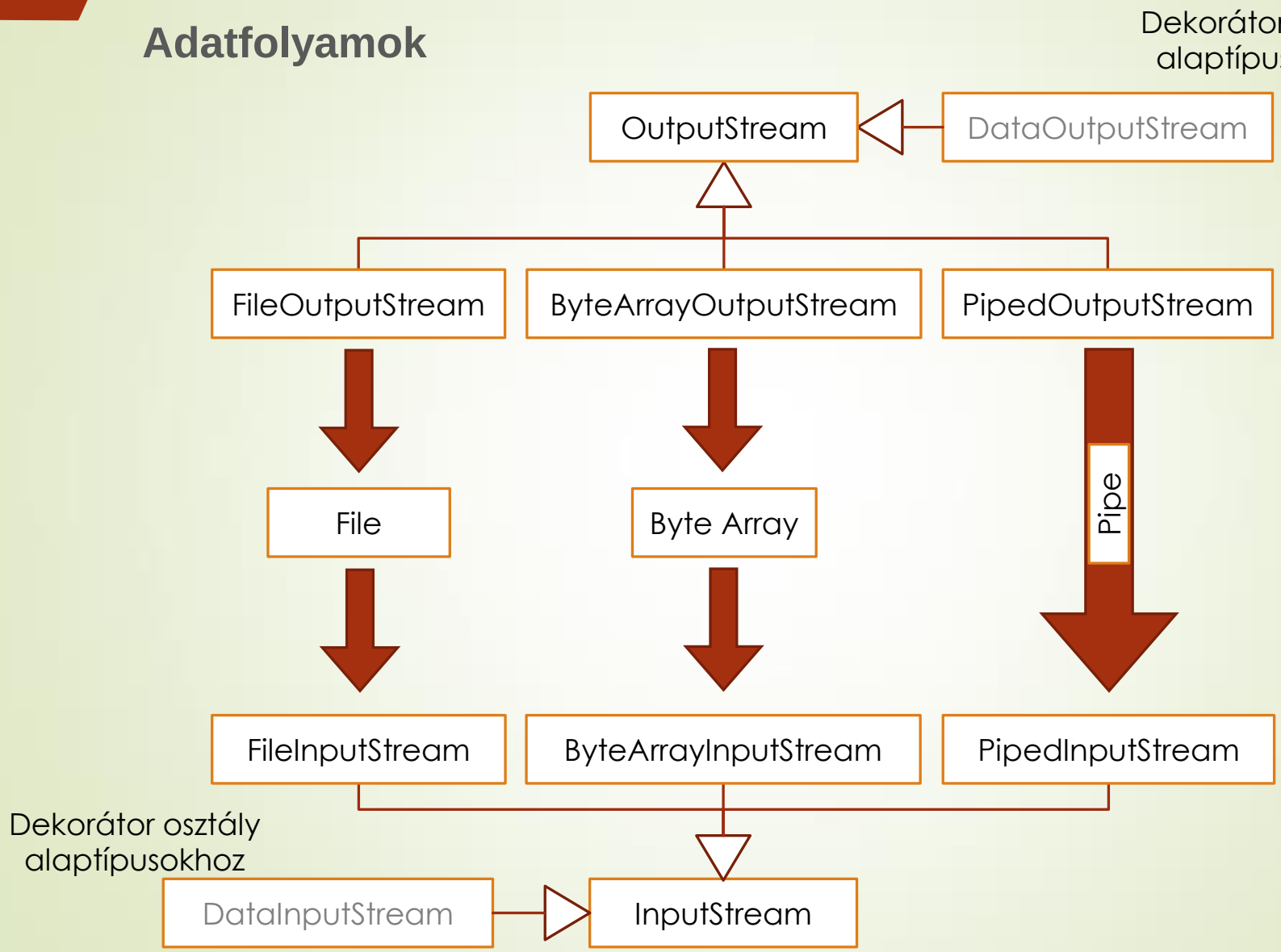
Hálózatkezelés

PacMan – A kliens és a szerver tesztelése

- A klienst és a szervert először hálózati kapcsolat használata nélkül implementáltuk, hogy lássuk működnek-e.
- Ahhoz, hogy a kapcsolatukat tesztelhessük, minden egyes kliens esetében a következőt kell tennünk
  - Kössük össze a kliens OutputStream-jét a szervernek a klienshez tartozó InputStream-jével.
  - Kössük össze a kliens InputStream-jét a szervernek a klienshez tartozó OutputStream-jével.
- Két kliens esetén a fentiek alapján 4 db egy irányú kapcsolatot kell kapnunk.
- Java-ban ehhez a PipedInputStream, PipedOutputStream használható, melyek közvetlenül összeköthetőek, egy csővezetéket létrehozva.

Hálózatkezelés

Adatfolyamok



Hálózatkezelés

PacMan – A kliens és a szerver tesztelése

```
PipedInputStream[]  client_in  = new PipedInputStream[2];
PipedInputStream[]  server_in  = new PipedInputStream[2];
PipedOutputStream[] client_out = new PipedOutputStream[2];
PipedOutputStream[] server_out = new PipedOutputStream[2];
GameBoardClient[]   clients    = new GameBoardClient[2];

for (int i = 0; i < 2; i++){
  client_in[i]  = new PipedInputStream();
  client_out[i] = new PipedOutputStream();
  server_in[i]  = new PipedInputStream(client_out[i]);
  server_out[i] = new PipedOutputStream(client_in[i]);
}

GameBoardServer server = new GameBoardServer(200, server_in, server_out);
server.start();

// Clients has to start AFTER the server is started, otherwise their
// constructor will block on their InputStream
for (int i = 0; i < 2; i++){
  clients[i] = new GameBoardClient(client_in[i], client_out[i], null);
  clients[i].start();
}

clients[0].changePlayerDirection(Direction.UP);
...

server.stop();
clients[0].stop();
clients[1].stop();
```

Hálózatkezelés

PacMan – Hálózati kapcsolat megvalósítása

- Az előző programfázisok sikeres tesztjei után most bővítsük ki a játékot úgy, hogy hálózatban is lehessen vele játszani.
- Tekintsük át először a hálózati kapcsolat alapfogalmait…

Hálózatkezelés

Alapok – InternetProtocol

- A hálózati kommunikációban résztvevő eszközöket az IP címük azonosítja (IPv4 vagy IPv6)
- Az IP (Internet Protocol) a forgalomirányításért felelős protokoll
- Egy eszközt több IP cím is azonosíthat
  - Önmagára hivatkozás: 127.0.0.1 (localhost, mindig létezik)
  - Internetes IP cím(ek)
- Az eszközön futó szolgáltatásokat a port szám azonosítja
  - Például:
    - HTTP: port 80
    - HTTPS: port 443
    - SSH: port 22

Hálózatkezelés

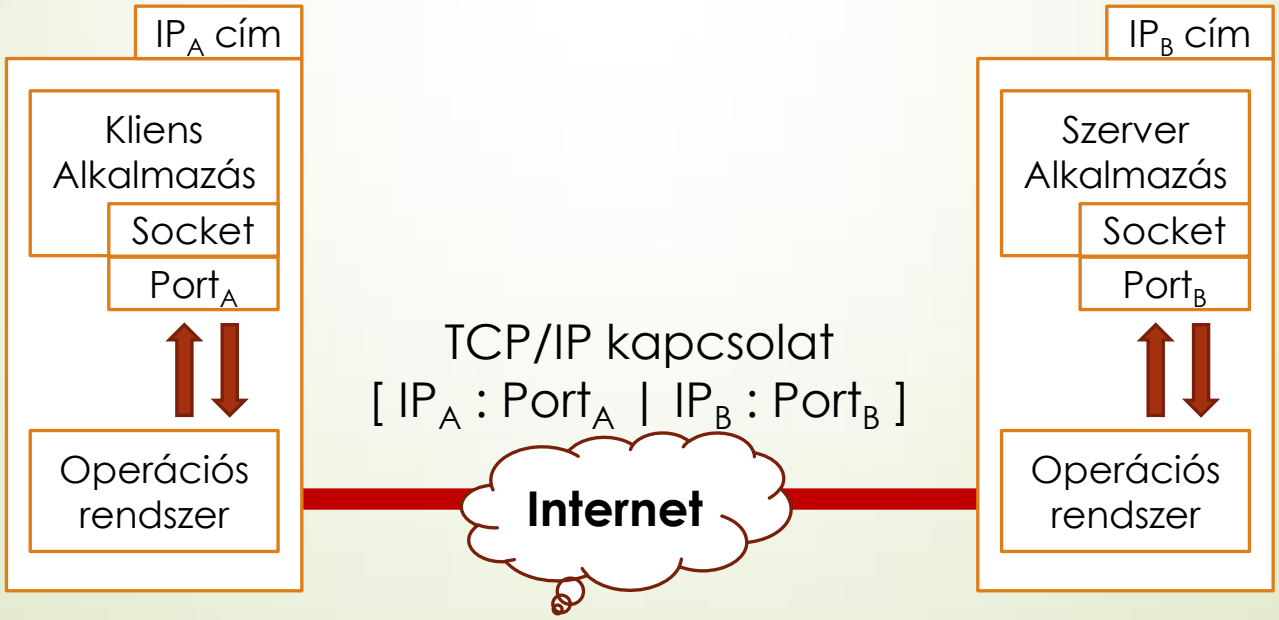
Alapok – adatátviteli protokollok

- A kommunikáció módja függ az alkalmazott protokolltól, például
  - TCP – Transfer Control Protocol
    - 1-1 kapcsolat, ami a kliens-szerver modellt implementálja
    - Kétirányú
    - Megbízható adatátvitel
  - UDP – Unified Datagram Protocol
    - Unicast – egy címzett
    - Multicast, Broadcast – több címzett
    - Egy irányú
    - Megbízhatatlan adatátvitel
    - Gyors kézbesítés

Hálózatkezelés

Alapok – TCP / IP kapcsolat

- Egy virtuális csővezeték, amit a két végpontja azonosít.
- A végpontot az (IP cím, port) páros azonosítja.
- A csomagról mindig ismert a forrás és cél IP címe, port száma.
- A 0..1023 port tartomány a szabvány szerinti szolgáltatásoknak fenntartott (pl. 53 – DNS, 80 – HTTP stb.).



Hálózatkezelés

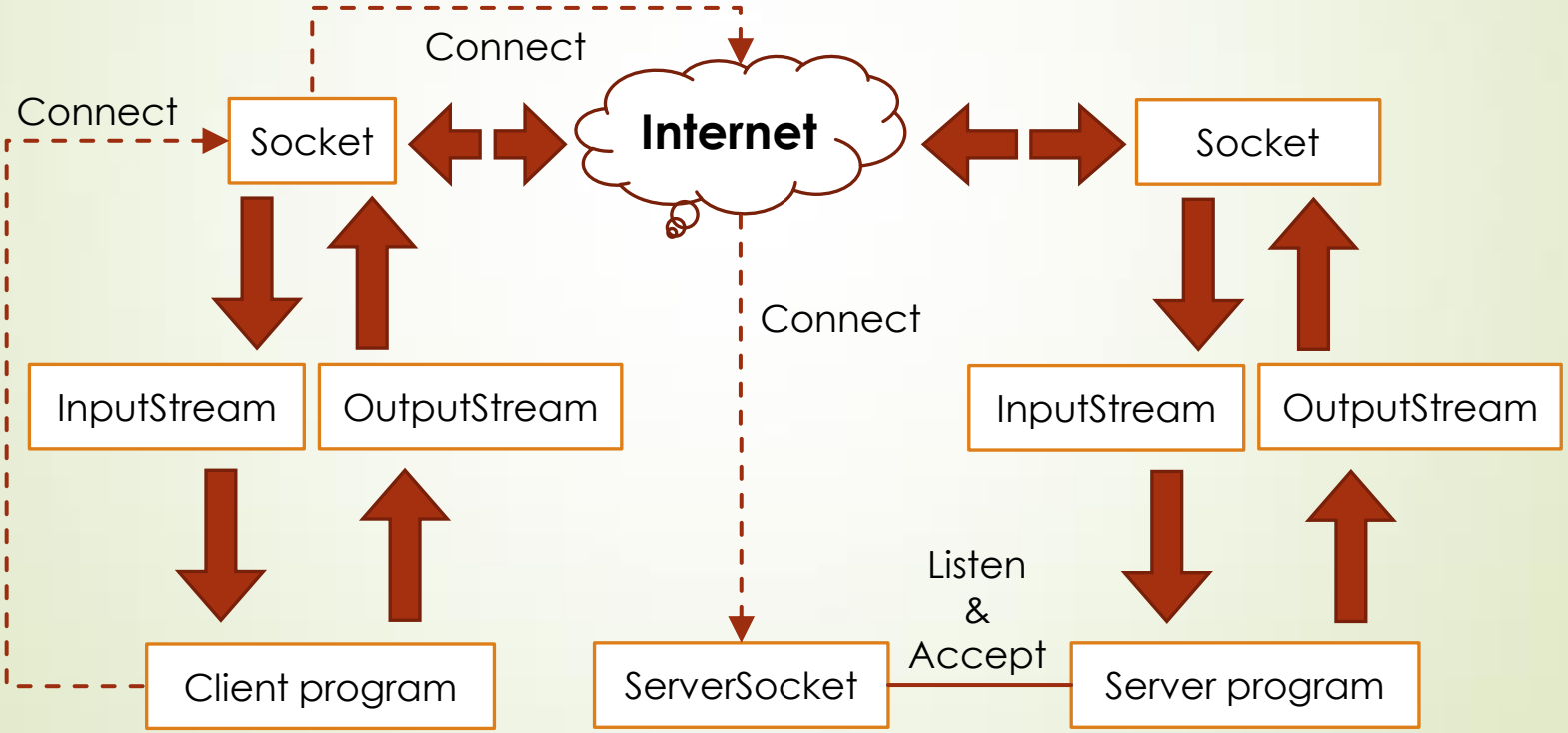
Alapok – TCP / IP kapcsolat felépítése

- Szerver alkalmazás
  - Egy általa választott, rá jellemző számú portot „nyit”, és ezen várja a kliensek kapcsolódását.
- Kliens alkalmazás
  - Az operációs rendszer segítségével kapcsolatot kezdeményez a szerverrel, annak IP cím és a port megadásával.
  - A kapcsolathoz az operációs rendszer oszt tetszőleges portot.

Hálózatkezelés

Hálózati kapcsolat Java-ban

- A socket az adatcsatorna szoftveres végpontja. Amit beleírunk, az megjelenik a másik oldalon. Java-ban a Socket-hez InputStream / OutputStream páros tartozik.



Hálózatkezelés

Hálózati kapcsolat Java-ban

- Java-ban két fajta Socket-et használunk
  - Socket
    - Az adatátvitel megvalósítására használható végpont, mind a kliens, mind pedig a szerver oldalon.
  - ServerSocket
    - Figyeli a csatlakozási kéréseket
    - Adatátvitelben nem vesz részt
    - Csatlakozáskor az új kapcsolathoz új Socket-et készít
- Érdemes beállítani a hálózati műveletekhez lejárati időt a setSoTimeout(int ms) metódussal. Így, ha nem történik semmi ezen időn belül, akkor SocketTimeoutException-t kapunk. Szálkezelés esetén a megszakíthatóságot garantálhatjuk vele.

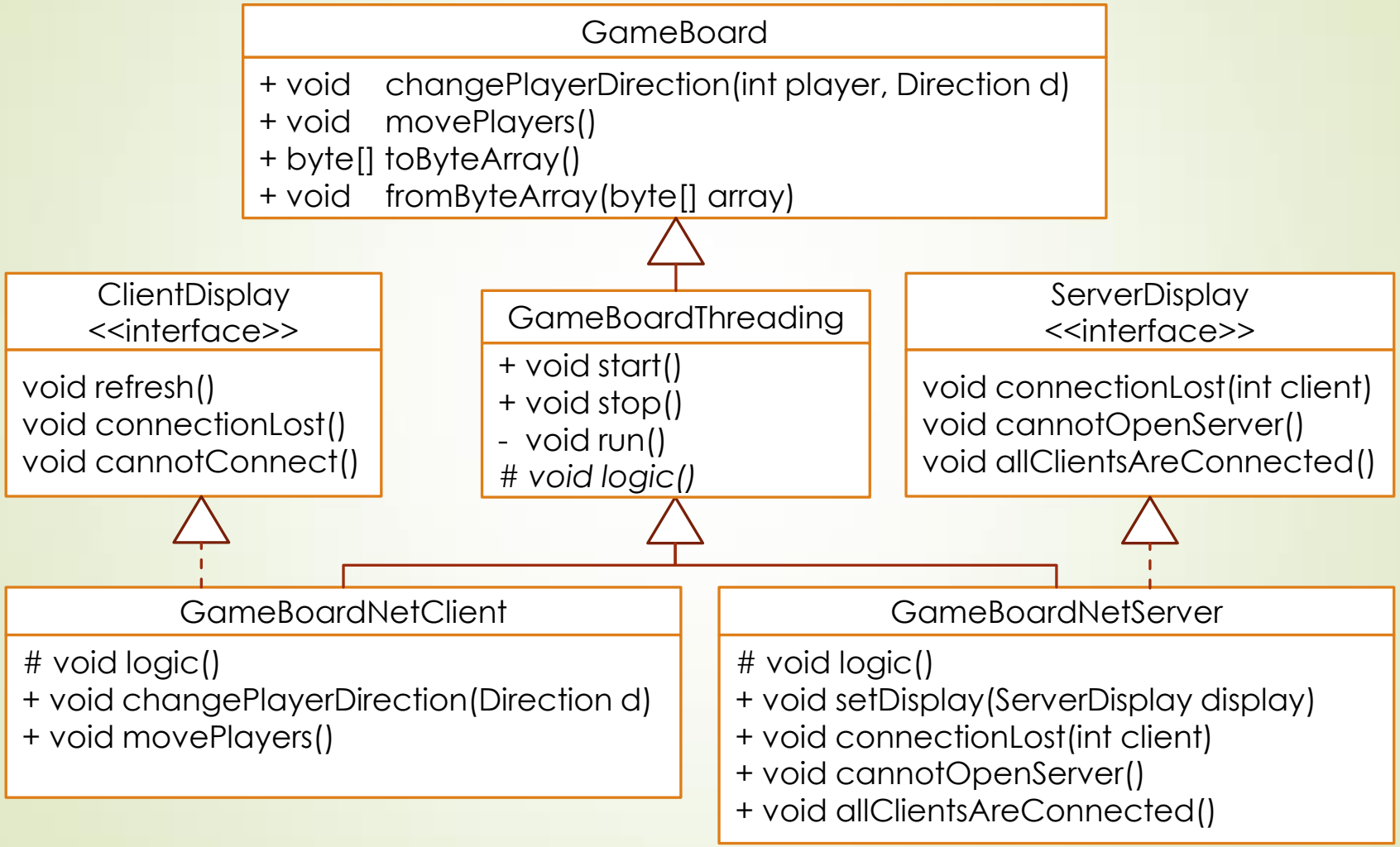
Hálózatkezelés

PacMan – Hálózatos kliens / szerver modellek

- Az előbbi információk birtokában készíthetünk egy a GameBoardClient / GameBoardServer-hez hasonló párost
- GameBoardNetClient
  - A megadott IP cím (vagy domain név) és port alapján kapcsolódik a szerverhez, majd a GameBoardClient-ben látottaknak megfelelően működik tovább.
- GameBoardNetServer
  - A megadott számú portot megnyitja, majd megvárja amíg az összes kliens kapcsolódik. Eközben már küld „Ping” jelzéseket (jelen esetben a modell bájt reprezentációját). A továbbiakban pontosan ugyanazt cselekszi, mint a GameBoardServer.

Hálózatkezelés

PacMan – Hálózatos kliens / szerver modellek



Hálózatkezelés

Tipp hálózatos alkalmazásokhoz

- Érdemes a „ping”-et a valós adattól megkülönböztetni. Ennek legegyszerűbb módja, ha a küldött adat első bájtja (int-je) meghatározza az adat típusát / célját (ping, frissítés…)
- Ez alapján tudhatjuk, hogy
  - él-e a kapcsolat, és
  - milyen típusú és mennyi adatot kell olvasni és feldolgozni.
- Példa
  - A kliensek az összes kliens kapcsolódásáig csak ping-eket kapnak, így látják, hogy a kapcsolat él, de a játék nem indult el.
- Hálózatos tesztelést gyakran végzünk egy gépen több futó alkalmazás példánnyal. Nehézség, hogy a billentyű és egér eseményeket a JRE csak az aktív ablaknak továbbítja. Ennek áthidalására a JNativeHook könyvtár használható.