Beszélgetőprogramok

Connection

Tekintsük a `Connection` osztályt, melynek segítségével a számítógépünkön futó Java Virtuális Gépek (és a bennük futó programok) kommunikálhatnak egymással. A kommunikáció során két fél között egy `Connection` objektumot használunk. A kommunikációban részt vevő felek közül az egyik az úgynevezett *szerver*, a másik pedig a *kliens*. Először mindig a szerver hozza létre a kapcsolatot, ezután a kliens automatikusan tud kapcsolódni hozzá.

A `Connection` osztály műveletei:

```
1 public static Connection accept() throws IOException
```

A szerver ezzel az osztályszintű művelettel hoz létre egy kapcsolatot. Hívása tehát: `Connection connection = Connection.accept()`.

```
1 public static Connection connect() throws IOException
```

A kliens ezzel az osztályszintű művelettel csatlakozik a már futó szerverhez. Hívása tehát: `Connection connection = Connection.connect()`.

```
1 public void send( String str ) throws IOException
```

Egy kiépített kapcsolaton keresztül küldhetünk egy szöveges üzenetet.

```
1 public String receive() throws IOException
```

Egy kiépített kapcsolaton keresztül fogadhatunk egy szöveges üzenetet. Ez egy blokkoló művelet: amíg nem kapunk üzenetet, a hívó folyamatot feltartja.

```
1 public void close() throws IOException
```

A kommunikáció végén mind a szerver, mind a kliens meg kell hívja ezt a műveletet. Mivel a `Connection` osztály megvalósítja az `AutoCloseable` interfészt, használhatjuk *try-with-resources* utasításban. A `close` művelet idempotens, azaz többször is meghívható.

--

Tekintsük most példaként a `Server` és `Client` osztályokat, melyek rendelkeznek `main` metódussal, azaz lefuttathatók egy-egy virtuális gépben.

Server

A szerverprogram elindít egy kapcsolatot, majd végtelen ciklusban fogad a kapcsolaton szöveges üzeneteket, melyeket nagybetűsít, majd visszaküld. A kapcsolat megszakadását `IOException` kivétel jelzi.

```
1  public class Server {
2    public static void main( String[] args ) {
3      try( Connection connection = Connection.accept() ){
4        while( true ){
5          connection.send( connection.receive().toUpperCase() );
6        }
7      } catch( java.io.IOException  e ){
8        System.err.println("Connection is lost.");
9      }
10   }
11 }
```

Kliens

A kliensprogramot a szerver után indítsuk el egy másik virtuális gépben. Ez a program megpróbál csatlakozni a szerverhez, majd sorra elküldi a szervernek a parancssori argumentumait. A visszaérkező válaszokat pedig kiírja a szabványos kimenetre.

```
1  public class Client {
2    public static void main( String[] args ) throws java.io.IOException {
3      try( Connection connection = Connection.connect() ){
4        for( int i=0; i<args.length; ++i ){
5          connection.send(args[i]);
6          System.out.println( connection.receive() );
7        }
8      }
9    }
10 }
```

Feladat

Készítsünk egy két szálon futó programot a fenti `Server` és `Client` mintájára, mely programmal két virtuális gépben futó program beszélgethet egymással. A programunk lényegi része legyen a `Talk` osztályban megvalósítva. E mellett az osztály mellett készítsünk egy `TalkServer` és egy `TalkClient` osztályt is, amelyek a `Talk` viselkedést szerverként, illetve kliensként elindítják. (Ezeket fogjuk majd programként külön JVM-ekben futtatni.) Segédosztályként készítsük el az `Alive` osztályt is, melyet a programunkat alkotó szálak leállításához használunk majd. (Lásd lentebb...)

```
1  class TalkServer {
2    public static void main( String[] args ){
3      Talk.communicate(true);
4    }
5  }
6
7  class TalkClient {
8    public static void main( String[] args ){
9      Talk.communicate(false);
10   }
11 }
12
13 class Talk {
14
15   private static void readFromConsole( Connection connection, Alive isAlive ){
16     // Ciklusban olvasunk a System.console-ról a readLine művelettel.
17     // Amit olvasunk, azt elküldjük a kapcsolaton keresztül.
18     // Leállás, amikor az isAlive hamisra vált,
19     // amikor a kapcsolatra írás kivételt vált ki,
20     // illetve amikor a readLine null-t olvas (EOF, pl. Ctrl-D).
21     // Leálláskor meghívjuk a shutdownn metódust.
22   }
23
24   private static void receiveFromConnection( Connection connection, Alive isAlive
   ){
25     // Ciklusban olvasunk a kapcsolatról.
26     // Amit olvasunk, azt kiírjuk a szabványos kimenetre.
27     // Leállás, amikor az isAlive hamisra vált,
28     // illetve amikor a kapcsolatról olvasás kivételt vált ki.
29     // Leálláskor meghívjuk a shutdownn metódust.
30   }
31
32   private static void shutdown( Connection connection, Alive isAlive ){
33     // Bezárjuk a kapcsolatot, és az isAlive-ot hamisra állítjuk.
34   }
35
36   public static void communicate( boolean isServer ){
37     // A kapott flagnek megfelelően létrehoz egy Connectiont.
38     // Létrehoz egy Alive objektumot is.
39     // Elindít egy szálat, mely a readFromConsole eljárást hajtja végre.
```

```
    // Ezután végrehajtja a receiveFromConnection eljárást.
  }

  private static class Alive {
    // Nyilvántart egy logikai értéket, mely kezdetben igaz.
    // Ezt az értéket a get() metódussal le lehet kérdezni.
    // A stop() metódussal az értéket hamisra lehet állítani.
    // Mivel az osztályt több szálból használjuk, legyen szálbiztos.
  }

}
```