Networking

TCP\IP in Java

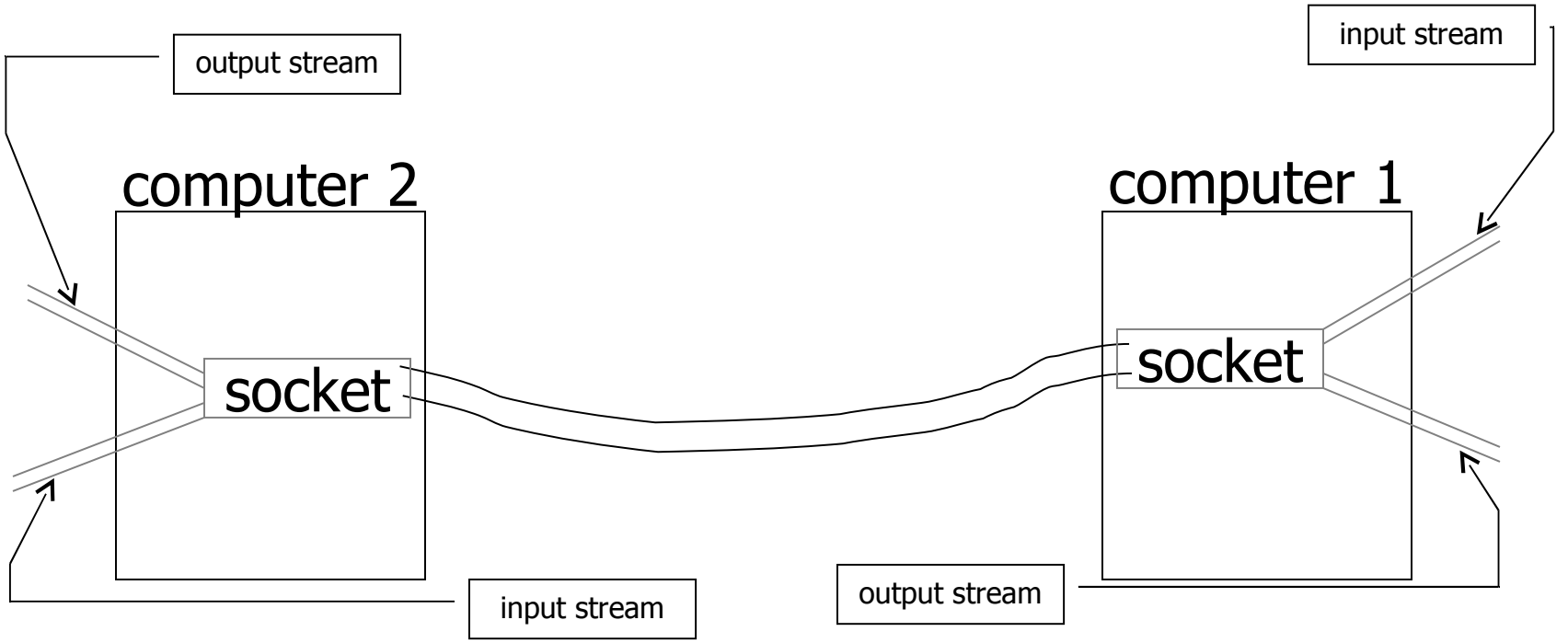
- The socket represents a little gate in the computer from which it is possible sending\receiving bytes to\from another computer that also has a socket.
- The socket object is instantiated from the Socket class.

TCP\IP in Java

- The socket object has two streams. One for sending bytes to an other socket on an other computer and one for receiving bytes from the other computer.
- The other socket can be on any other computer that has network connection with the current computer.
- Once the two computers have sockets on each one of them, each one of the two sockets can send and get information to\from the other.

TCP\IP in Java

- Having two computers connected via sockets through which they send/receive data with each other can be described as follows:



Connection Establishment

- The server takes the first step instantiating the ServerSocket class.
  ServerSocket serverSocket = new ServerSocket(1300);
- The server calls the accept() method on the server socket object. The accept() method makes the server socket object waiting for a request (from the client) to make the connection.
  Socket socket = serverSocket.accept();
- When a request to make a connection arrives from a client, the accept method succeeds in its job and returns a reference to socket.

Connection Establishment

- The client tries to instantiate the Socket class. It sends the Socket class constructor two arguments: the name of the computer on which the server runs and the port number to which the server listens.
  Socket socket = new Socket(“http://www.zindell.co.il”,1300);
- When the constructor succeeds it means that a connection was established and the two sockets are connected.

Connection Establishment

- The server moves the socket from the port it listened to, and use another port instead. This allows the server getting more requests for connection, concurrently with other active connections.
- Now, both sides, the client and the server, can call the getOutputStream() and getInputStream() methods on their sockets.

Connection Establishment

- Now, each side can send and receive data.
- It is common to decide on agreed protocol for the communication between the two.
- In the end of the connection, both the client and the server should invoke the close() method on the socket objects as well as on each one of the streams.

Simple Client

```
import java.net.*;
import java.io.*;

public class SimpleClient
{
  public static String serverName = "127.0.0.1";
  public static int serverPortNumber = 1300;

  public static void main(String args[])
  {
    Socket socket = null;
    InputStream is = null;
    DataInputStream dis = null;
    OutputStream os = null;
    DataOutputStream dos = null;
```

Simple Client

```
try
{
  socket = new
  Socket(serverName,serverPortNumber);
  System.out.println("socket was created...");
  is = socket.getInputStream();
  System.out.println("input stream was created...");
  dis = new DataInputStream(is);
  System.out.println("data input stream was created...");
  os = socket.getOutputStream();
  System.out.println("output stream was created...");
  dos = new DataOutputStream(os);
```

Simple Client

```
  System.out.println("data output stream was created...");
  dos.writeUTF(args[0]);
  System.out.println(args[0]+" was written to server...");
  System.out.println(dis.readUTF() + " was received...");
}
catch(IOException e)
{
   e.printStackTrace();
}
```

Simple Client

```
finally
{
  if(is!=null)
  {
    try{is.close();}catch(IOException e){e.printStackTrace();}
  }
  if(os!=null)
  {
    try{os.close();}catch(IOException e){e.printStackTrace();}
  }
  if(dis!=null)
  {
    try{dis.close();}catch(IOException e){e.printStackTrace();}
  }
```

Simple Client

```
      if(dos!=null)
      {
        try{dos.close();}catch(IOException e){e.printStackTrace();}
      }
      if(socket!=null)
      {
        try{socket.close();}catch(IOException e){e.printStackTrace();}
      }
    } //finally
  } //main
} //class
```

Simple Server

```
import java.net.*;
import java.io.*;

public class SimpleServer
{
  public static String serverName = "127.0.0.1";
  public static int serverPortNumber = 1300;
  public static void main(String args[])
  {
    Socket socket = null;
    ServerSocket server = null;
    InputStream is = null;
    DataInputStream dis = null;
    OutputStream os = null;
    DataOutputStream dos = null;
```

Simple Server

```
try
{
  server = new ServerSocket(serverPortNumber);
  while(true)
  {
    System.out.println("server.accept()...");
    socket = server.accept();
    is = socket.getInputStream();
    dis = new DataInputStream(is);
    os = socket.getOutputStream();
    dos = new DataOutputStream(os);
    String number = dis.readUTF();
    System.out.println("dis.readUTF() returns "+number);
    String reply = "You should send the server one of the
    following numbers: 1,2 or 3";
    if(number.equals("1")) {reply = "one";}
    else if(number.equals("2")) {reply = "two";}
    else if(number.equals("3")) {reply = "three";}
```

Simple Server

```
      System.out.println("dos.writeUTF(\""+reply+"\")");
      dos.writeUTF(reply);
  }
}
catch(IOException e)
{
  e.printStackTrace();
}
finally
{
  if(is!=null)
  {
    try{is.close();}catch(IOException e){e.printStackTrace();}
  }
  if(os!=null)
  {
    try{os.close();}catch(IOException e){e.printStackTrace();}
  }
```

Simple Server

```
      if(dis!=null)
      {
        try{dis.close();}catch(IOException e){e.printStackTrace();}
      }
      if(dos!=null)
      {
        try{dos.close();}catch(IOException e){e.printStackTrace();}
      }
      if(socket!=null)
      {
        try{socket.close();}catch(IOException e){e.printStackTrace();}
      }
    }
  }
}
```

Networking

TCP\IP in Java

- The socket represents a little gate in the computer from which it is possible sending\receiving bytes to\from another computer that also has a socket.
- The socket object is instantiated from the Socket class.

TCP\IP in Java

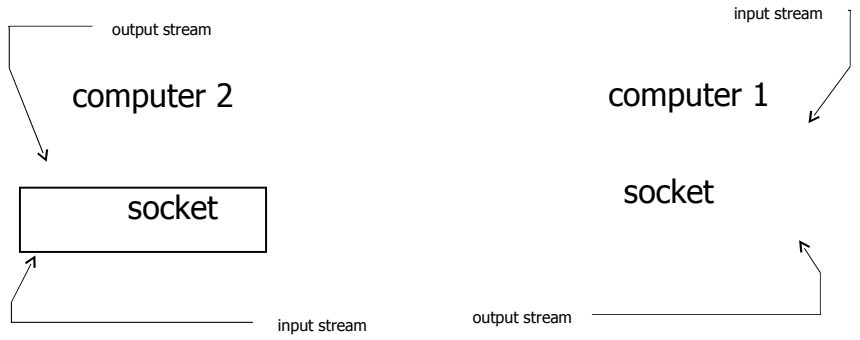
- The socket object has two streams. One for sending bytes to an other socket on an other computer and one for receiving bytes from the other computer.
- The other socket can be on any other computer that has network connection with the current computer.
- Once the two computers have sockets on each one of them, each one of the two sockets can send and get information to\from the other.

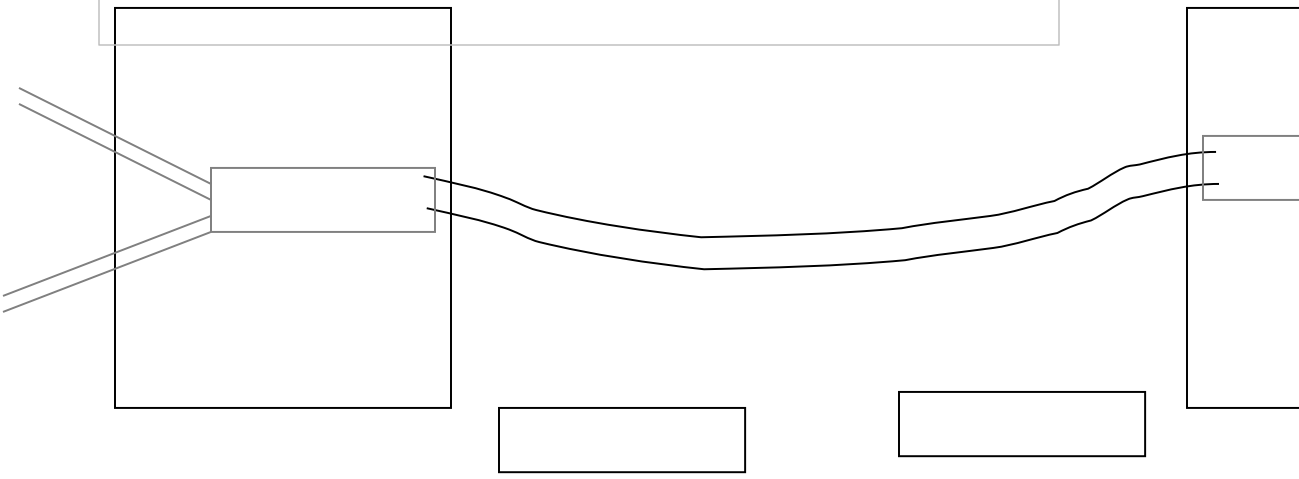
If you know the Stargate sci-fi series, the concept of having a socket through which it is possible to send/receive data from one computer to another should look very similar to the concept the star-gate is based on.

If you are not familiar with Stargate SG-1 and with Stargate Atlantis I strongly recommend you to check these two excellent TV shows!

TCP\IP in Java

- Having two computers connected via sockets through which they send/receive data with each other can be described as follows:





11:45 AM Abelski eLearning©

Connection Establishment

- The server takes the first step instantiating the ServerSocket class.
  ServerSocket serverSocket = new ServerSocket(1300);
- The server calls the accept() method on the server socket object. The accept() method makes the server socket object waiting for a request (from the client) to make the connection.
  Socket socket = serverSocket.accept();
- When a request to make a connection arrives from a client, the accept method succeeds in its job and returns a reference to socket.

The port numbers in TCP\IP systems are 16-bit numbers and range from 0 to 65535. In practice, port numbers below 1024 are reserved for predefined services (email, ftp, http etc…). Unless the intention is to communicate with one of those services these ports shouldn't be used.

The ServerSocket constructor receives the port number which he should listen.

11:45 AM Abelski eLearning©

Connection Establishment

- The client tries to instantiate the Socket class. It sends the Socket class constructor two arguments: the name of the computer on which the server runs and the port number to which the server listens.
  Socket socket = new Socket("http://www.zindell.co.il",1300);
- When the constructor succeeds it means that a connection was established and the two sockets are connected.

When the socket returns InputStream and OutputStream references, it is possible to connect them with other streams in order to have more functional streams (e.g. connecting them with DataInputStream & DataOutputStream).

Connection Establishment

- The server moves the socket from the port it listened to, and use another port instead. This allows the server getting more requests for connection, concurrently with other active connections.
- Now, both sides, the client and the server, can call the getOutputStream() and getInputStream() methods on their sockets.

Connection Establishment

- Now, each side can send and receive data.
- It is common to decide on agreed protocol for the communication between the two.
- In the end of the connection, both the client and the server should invoke the close() method on the socket objects as well as on each one of the streams.

Simple Client

```
import java.net.*;
import java.io.*;

public class SimpleClient
{
  public static String serverName = "127.0.0.1";
  public static int serverPortNumber = 1300;

  public static void main(String args[])
  {
    Socket socket = null;
    InputStream is = null;
    DataInputStream dis = null;
    OutputStream os = null;
    DataOutputStream dos = null;
```

Simple Client

```
try
{
  socket = new
  Socket(serverName,serverPortNumber);
  System.out.println("socket was created...");
  is = socket.getInputStream();
  System.out.println("input stream was created...");
  dis = new DataInputStream(is);
  System.out.println("data input stream was created...");
  os = socket.getOutputStream();
  System.out.println("output stream was created...");
  dos = new DataOutputStream(os);
```

Simple Client

```
  System.out.println("data output stream was created...");
  dos.writeUTF(args[0]);
  System.out.println(args[0]+" was written to server...");
  System.out.println(dis.readUTF() + " was received...");
}
catch(IOException e)
{
   e.printStackTrace();
}
```

Simple Client

```
finally
{
  if(is!=null)
  {
    try{is.close();}catch(IOException e){e.printStackTrace();}
  }
  if(os!=null)
  {
    try{os.close();}catch(IOException e){e.printStackTrace();}
  }
  if(dis!=null)
  {
    try{dis.close();}catch(IOException e){e.printStackTrace();}
  }
```

Simple Client

```
      if(dos!=null)
      {
        try{dos.close();}catch(IOException e){e.printStackTrace();}
      }
      if(socket!=null)
      {
        try{socket.close();}catch(IOException e){e.printStackTrace();}
      }
    } //finally
  } //main
} //class
```

Simple Server

```
import java.net.*;
import java.io.*;

public class SimpleServer
{
  public static String serverName = "127.0.0.1";
  public static int serverPortNumber = 1300;
  public static void main(String args[])
  {
    Socket socket = null;
    ServerSocket server = null;
    InputStream is = null;
    DataInputStream dis = null;
    OutputStream os = null;
    DataOutputStream dos = null;
```

Simple Server

```
try
{
  server = new ServerSocket(serverPortNumber);
  while(true)
  {
    System.out.println("server.accept()...");
    socket = server.accept();
    is = socket.getInputStream();
    dis = new DataInputStream(is);
    os = socket.getOutputStream();
    dos = new DataOutputStream(os);
    String number = dis.readUTF();
    System.out.println("dis.readUTF() returns "+number);
    String reply = "You should send the server one of the
    following numbers: 1,2 or 3";
    if(number.equals("1")) {reply = "one";}
    else if(number.equals("2")) {reply = "two";}
    else if(number.equals("3")) {reply = "three";}
```

Simple Server

```
      System.out.println("dos.writeUTF(\""+reply+"\")");
      dos.writeUTF(reply);
    }
  }
  catch(IOException e)
  {
    e.printStackTrace();
  }
  finally
  {
    if(is!=null)
    {
      try{is.close();}catch(IOException e){e.printStackTrace();}
    }
    if(os!=null)
    {
      try{os.close();}catch(IOException e){e.printStackTrace();}
    }
```

Simple Server

```
      if(dis!=null)
      {
        try{dis.close();}catch(IOException e){e.printStackTrace();}
      }
      if(dos!=null)
      {
        try{dos.close();}catch(IOException e){e.printStackTrace();}
      }
      if(socket!=null)
      {
        try{socket.close();}catch(IOException e){e.printStackTrace();}
      }
    }
  }
}
```