Advanced Java Programming

Part 9

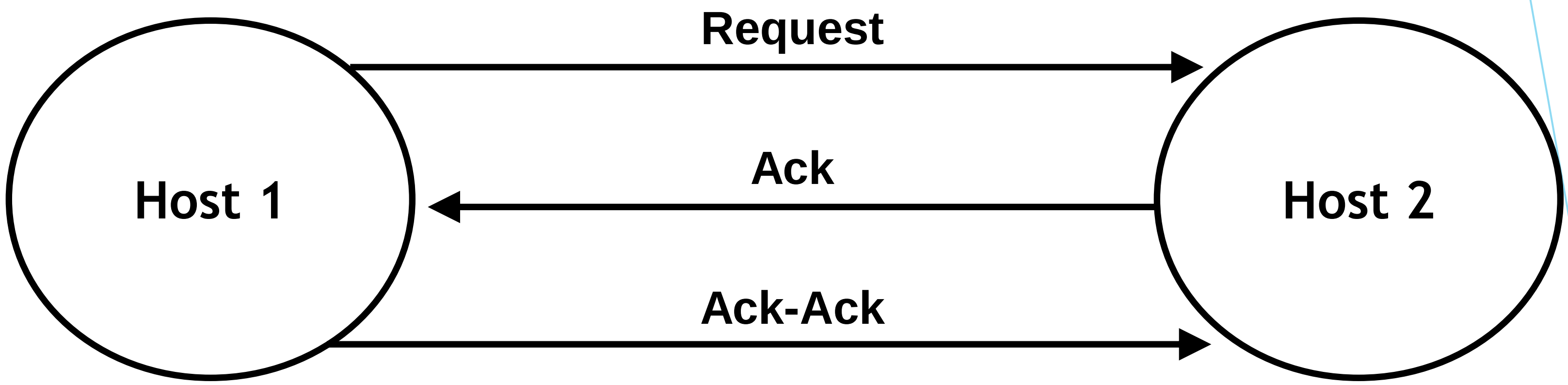
Presented By
Eng : Hager Samir

Networking in Java

Overview of TCP and UDP Protocols

TCP	UDP
Connection Oriented (Handshaking procedure)	Connection Less
Continuous Stream	Message Oriented
Reliable (Error Detection)	Unreliable

3-Way Handshaking Procedure



Establishing a Connection

- In order to connect to a remote host, two pieces of information are essentially required:
  - IP Address (of remote machine)
  - Port Number (to identify the service at the remote machine)

- **Socket = Address + Port**

- Range of port numbers: 0 → 65,535

- From 0 → 1,024 are reserved for well known services, such as:
  - HTTP: 80
  - FTP: 21
  - Telnet: 23
  - SMTP: 25

Basic Client/Server Communication

Server	Client
1. Create a server socket (bind the service to a certain port)	
2. Listen for connections	1. Create a socket (connect to the server)
3. Accept connection and transfer the client request to a virtual port.	
4. Obtain input and output streams	2. Obtain input and output streams
5. Send and receive data	3. Send and receive data.
6. Terminate connection (after communication has ended)	4. Terminate connection (after communication has ended)

ServerSocket Class

- Commonly Used Constructor(s):
  - `ServerSocket(int port)`
  - `ServerSocket(int port, int maxCon)`

- Commonly Used Method(s):
  - `Socket accept()`
  - `close()`

Socket Class

- Commonly Used Constructor(s):
  - `Socket(String address, int port)`
  - `Socket(InetAddress address, int port)`

- Commonly Used Method(s):
  - `InputStream getInputStream()`
  - `OutputStream getOutputStream()`

InetAddress Class

- InetAddress class has no public constructor.

- Commonly Used Method(s):
  - `static InetAddress getByName(String host)`
  - `static InetAddress[] getAllByName(String host)`
  - `static InetAddress getLocalHost()`
  - `String getHostName()`
  - `String getHostAddress()`
  - `Byte[] getAddress()`

Simple Client/Server Console Example
Server Application

- The following code sample is for creating a simple one-to-one client/server application, where each machine sends out a string and receives a string:

```
public class Server {
    ServerSocket myServerSocket;
    Socket waiter;
    DataInputStream dis ;
    PrintStream ps;
   public static void main(String[] args) {
          new Server();
    }
   public Server() {
       try {
             myServerSocket = new ServerSocket(5005);
             waiter = myServerSocket.accept ();
            dis = new DataInputStream(waiter.getInputStream ());
             ps = new PrintStream(waiter.getOutputStream ());
```

Simple Client/Server Console Example
Server Application cont’d

```
                String msg = dis.readLine();
                System.out.println(msg);
                ps.println("Data Received");
            }
            catch(IOException ex) {
                ex.printStackTrace();
            }
            finally {
                try {
                ps.close();
                    dis.close();
                    s.close();
                    myServerSocket.close();
                }
                catch(Exception ex) {
                    ex.printStackTrace();
                }
            }
        }
    }
```

Simple Client/Server Console Example
Client Application

```
public class Client
{
    Socket mySocket;
    DataInputStream dis ;
    PrintStream ps;
    public static void main(String[] args) {
          new Client();
    }

    public Client() {
        try {
            mySocket = new Socket("127.0.0.1", 5005);
            dis = new DataInputStream(mySocket.getInputStream ());
            ps = new PrintStream(mySocket.getOutputStream ());
            ps.println("Test Test");
            String replyMsg = dis.readLine();
            System.out.println(replyMsg);
        }
```

Simple Client/Server Console Example
Client Application cont’d

```
            catch(IOException ex) {
                ex.printStackTrace();
            }
            finally {
                try {
                 ps.close();
                 dis.close();
                 mySocket.close();
                 }
                catch(Exception ex) {
                    ex.printStackTrace();
                }
            }
        }
    }
```

Lab

Simple Client Server CMD Application

- Develop a client/server command line applications where
- the client:
  - Can send multiple messages to the server by typing them on the command line then pressing Enter Key
  - It can exit by letting the user type “exit” then press enter
- The Server
  - Always receives the messages from the client and prints them to on his console.
  - If received an “exit” message, it will exit