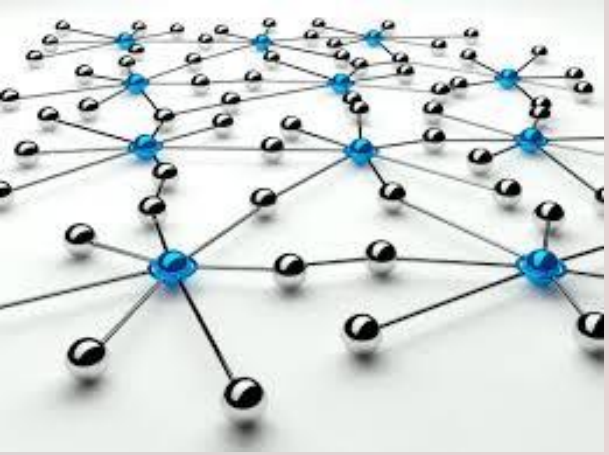
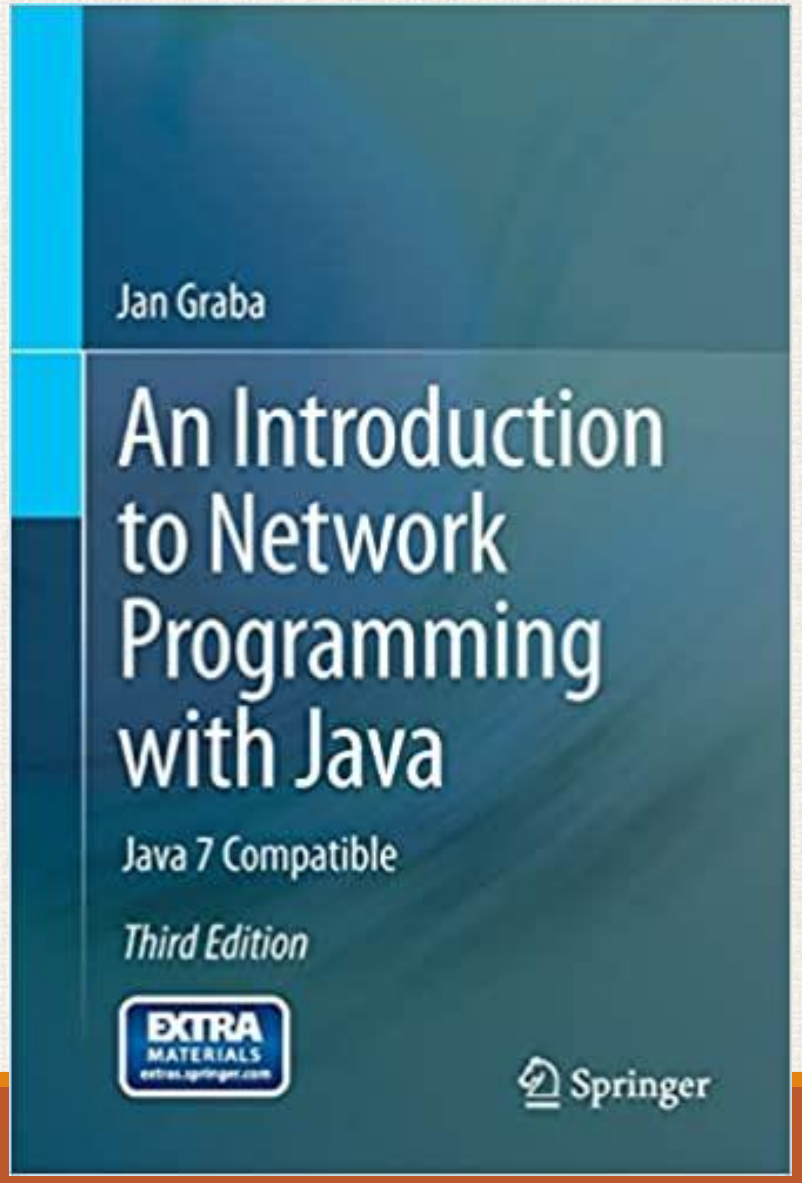


Network Programming with Java

GANESH PAI
ASST. PROFESSOR GD III
DEPARTMENT OF CSE
NMAMIT, NITTE


Java

Textbook


Jan Graba
An Introduction to Network Programming with Java
Java 7 Compatible
Third Edition
EXTRA MATERIALS
Springer

Overview of topics

- Clients, Servers and Peers
- The Internet
- IP Addresses, Port numbers
- Sockets
- Internet Services, URLs and DNS
- TCP & UDP
- InetAddress class
- Using TCP Sockets
- Using UDP Sockets

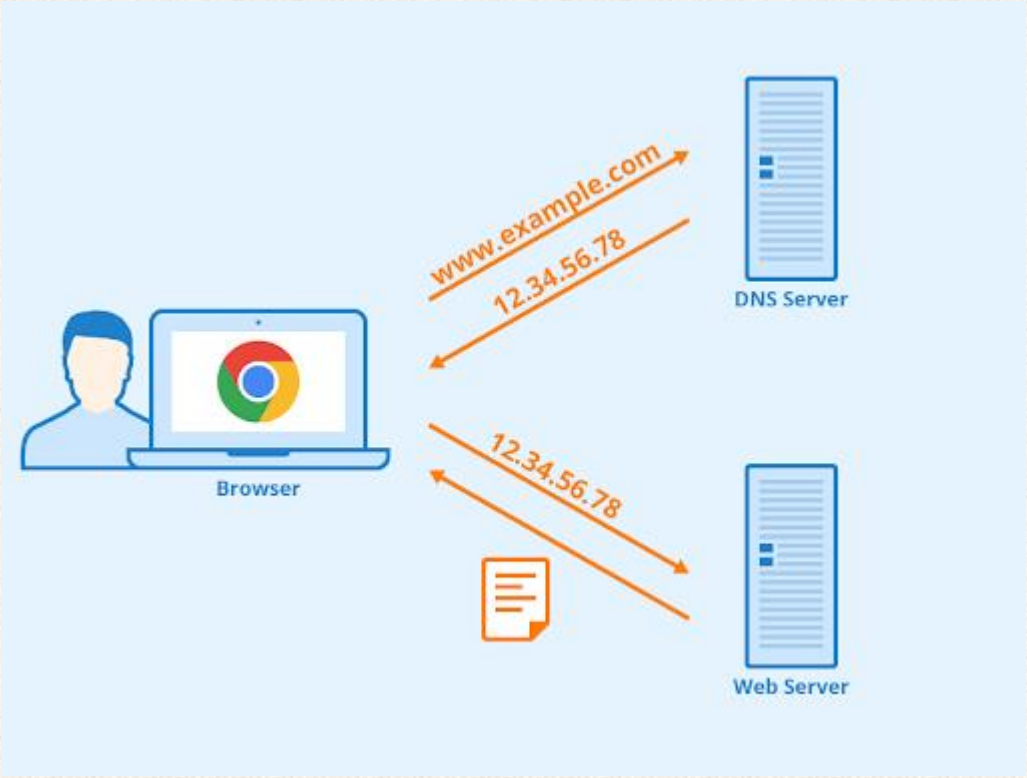
Internet Services

Protocol name	Port number	Nature of service
Echo	7	The server simply echoes the data sent to it. This is useful for testing purposes
Daytime	13	Provides the ASCII representation of the current date and time on the server
FTP-data	20	Transferring files. (FTP uses two ports.)
FTP	21	Sending FTP commands like PUT and GET
Telnet	23	Remote login and command line interaction
SMTP	25	E-mail. (Simple Mail Transfer Protocol.)
HTTP	80	HyperText Transfer Protocol (the World Wide Web protocol)
NNTP	119	Usenet. (Network News Transfer Protocol.)

URL (Uniform Resource Locator) Format & DNS

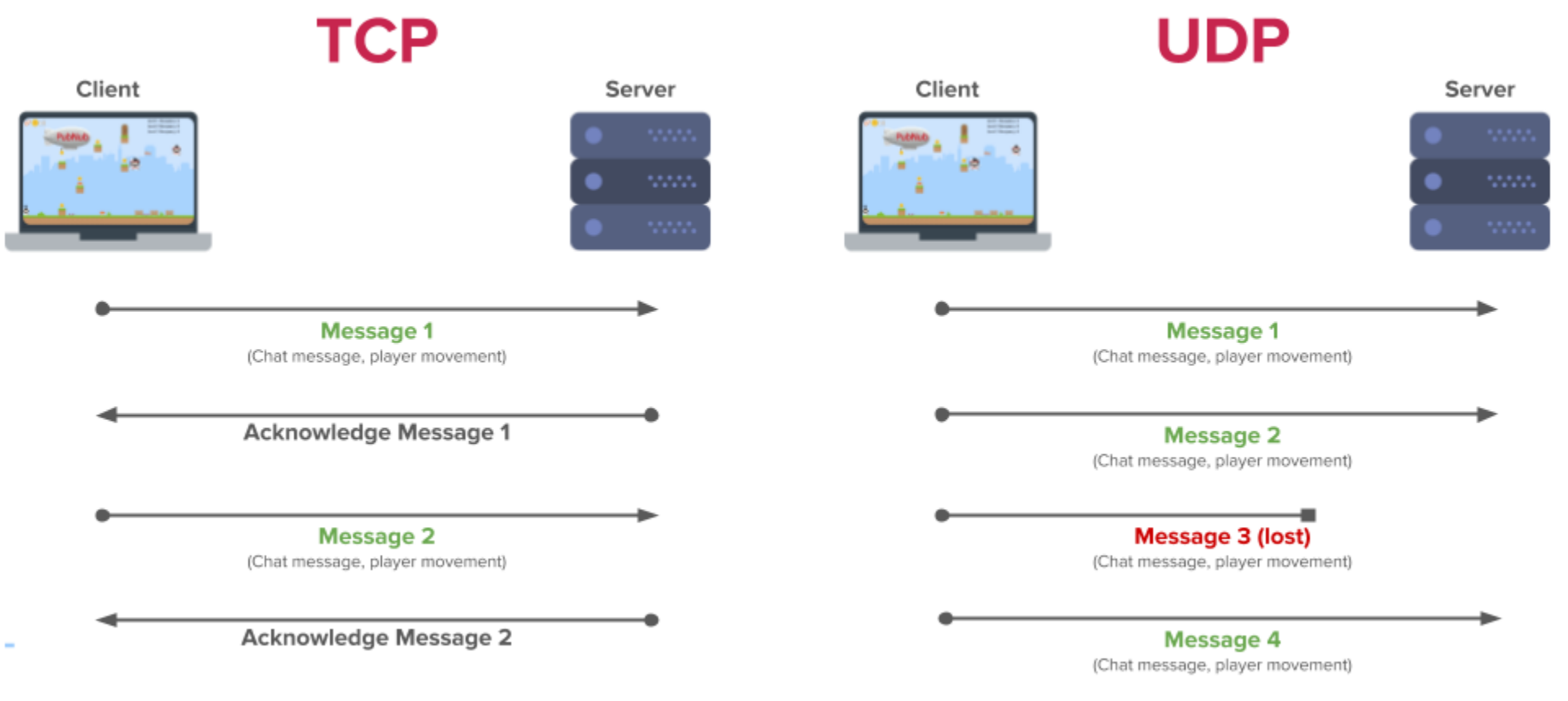
```
<protocol>://<hostname>[:<port>][/<pathname>] [/<filename>[#<section>]]
```

- http://www.oracle.com/technetwork/java/javase/downloads/index.html




Java

TCP & UDP


TCP
Client
Server
Message 1
(Chat message, player movement)
Acknowledge Message 1
Message 2
(Chat message, player movement)
Acknowledge Message 2
UDP
Client
Server
Message 1
(Chat message, player movement)
Message 2
(Chat message, player movement)
Message 3 (lost)
(Chat message, player movement)
Message 4
(Chat message, player movement)

InetAddress class

- Available in **java.net**
- Handles Internet addresses as host names and as IP addresses.
- Static method **getByName()** uses DNS (Domain Name System) to return the Internet address of a specified host name as an InetAddress object.
- **getByName()** throws the checked exception *UnknownHostException* if the host name is not recognized
- **InetAddress.getLocalHost()** returns IP address of current machine

```
import java.net.*;
import java.util.*;

public class IPFinder
{
     public static void main(String[] args)
     {
          String host;
          Scanner input = new Scanner(System.in);
          InetAddress address;

          System.out.print("\n\nEnter host name: ");
          host = input.next();
          try
          {
               address = InetAddress.getByName(host);
               System.out.println("IP address: "
                                  + address.toString());
          }
          catch (UnknownHostException uhEx)
          {
               System.out.println("Could not find " + host);
          }
     }
}
```

```
Enter host name: java.sun.com
IP address: java.sun.com/192.18.97.71
```

TCP Sockets - Steps

Setting up Server Socket

- 1. Create a ServerSocket object.

```
ServerSocket serverSocket = new ServerSocket(1234);
```

- 2. Put the server into a waiting state.

```
Socket link = serverSocket.accept();
```

- 3. Set up input and output streams.

```
Scanner input = new Scanner(link.getInputStream());
PrintWriter output = new PrintWriter(link.getOutputStream(),true);
```

- 4. Send and receive data.

```
output.println("Awaiting data…");
String input = input.nextLine();
```

- 5. Close the connection (after completion of the dialogue).

```
link.close();
```

TCP Sockets - Steps

Setting up Client Socket

- 1. Establish a connection to the server

```
Socket link = new Socket(SERVER_IP, SERVER_PORT);
```

- 2. Set up input and output streams

```
Scanner input = new Scanner(link.getInputStream());
PrintWriter output = new PrintWriter(link.getOutputStream(),true);
```

- 3. Send and receive data.

```
output.println("Awaiting data…");
String input = input.nextLine();
```

- 4. Close the connection.

```
link.close();
```

UDP Sockets - Steps

Setting up Server Socket

- 1. Create a DatagramSocket object

```
DatagramSocket datagramSocket = new DatagramSocket(1234);
```

- 2. Create a buffer for incoming datagrams

```
byte[] buffer = new byte[256];
```

- 3. Create a DatagramPacket object for the incoming datagrams

```
DatagramPacket inPacket = new DatagramPacket(buffer, buffer.length);
```

- 4. Accept an incoming datagram.

```
datagramSocket.receive(inPacket);
```

UDP Sockets - Steps

- 5. Accept the sender's address and port from the packet.

```
InetAddress clientAddress = inPacket.getAddress();
int clientPort = inPacket.getPort();
```

- 6. Retrieve the data from the buffer

```
String message = new String(inPacket.getData(), 0,inPacket.getLength());
```

- 7. Create the response datagram.

```
DatagramPacket outPacket = new DatagramPacket(response.getBytes(), response.length(),
                                                clientAddress, clientPort);
```

- 8. Send the response datagram.

```
datagramSocket.send(outPacket);
```

- 9. Close the DatagramSocket

```
datagramSocket.close();
```

UDP Sockets - Steps

Setting up Client Socket

- 1. Create a DatagramSocket object

```
DatagramSocket datagramSocket = new DatagramSocket(1234);
```

- 2. Create the outgoing datagram

```
DatagramPacket outPacket = new DatagramPacket(message.getBytes(), message.length(),
                                              host, PORT);
```

- 3. Send the datagram message

```
datagramSocket.send(outPacket);
```

- 4. Create a buffer for incoming datagrams

```
byte[] buffer = new byte[256];
```

UDP Sockets - Steps

- 5. Create a DatagramPacket object for the incoming datagrams

```
DatagramPacket inPacket = new DatagramPacket(buffer, buffer.length);
```

- 6. Accept an incoming datagram.

```
datagramSocket.receive(inPacket);
```

- 7. Retrieve the data from the buffer.

```
String response = new String(inPacket.getData(), 0, inPacket.getLength());
```

- 8. Close the DatagramSocket.

```
datagramSocket.close();
```