2017年度ネットワーク特論１
第10回・第12回

演習２課題イ

2017年5月17日（水）・24日

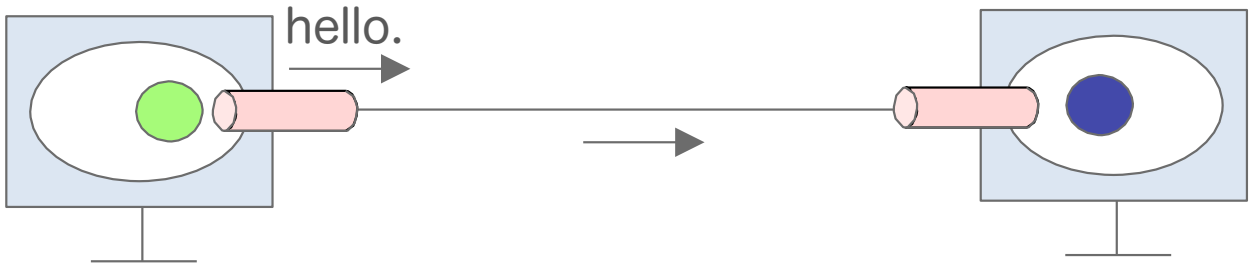
産業技術大学院大学

情報アーキテクチャ専攻

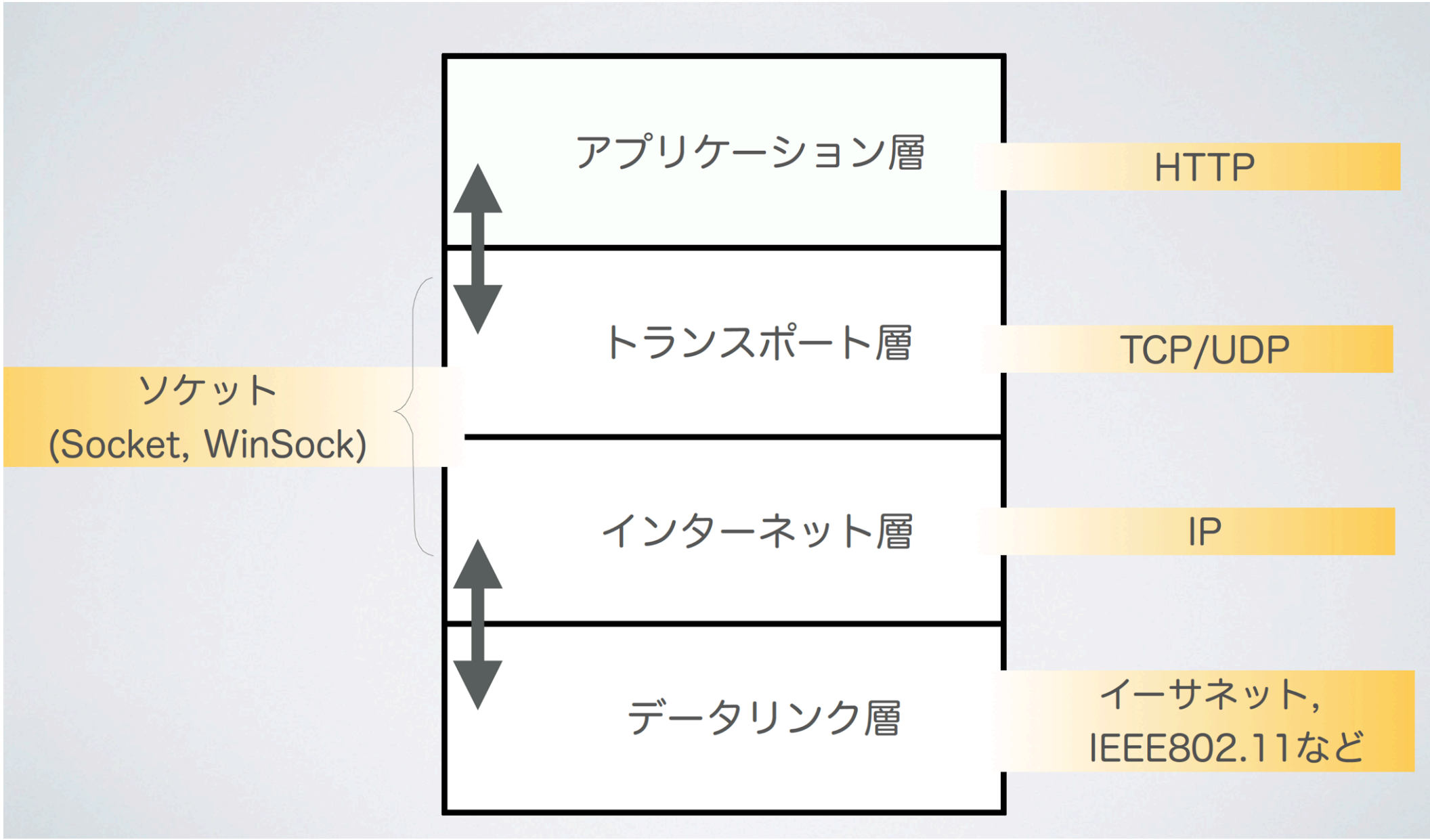
助教　　大﨑理乃

5.NWプログラミング

- ソケット
  - アプリケーションに対してプロセス間通信の環境を提供
  - IPアドレスとポート番号の組を「ソケット」と呼ぶ
  - プロトコルとしてTCPを使うかUDPを使うかを選ぶ必要がある
    - TCP：Socket、ServerSocket
    - UDP：DatagramPacket、DatagramSocket



プロトコルと階層構造


アプリケーション層
HTTP
トランスポート層
TCP/UDP
ソケット
(Socket, WinSock)
インターネット層
IP
データリンク層
イーサネット,
IEEE802.11など

準備（Javaの場合）

- Editor
  - Unix/Mac
    - vi
    - Emacs
  - Windows
    - 色々
- コンパイラー
  - JDK

NWプログラミング

- Javaでの実装例（サーバ）

```
/*
 * SimpleServer.java: 受け取ったメッセージをそのまま返すサーバ
 */
import java.io.*;
import java.net.*;

public class SimpleServer {
  public static final int PORT = 5555;          // ポート番号

  public static void main (String args[]) {

    try {                                                  // ソケットを生成
      ServerSocket serverSocket = new ServerSocket(PORT);

      // 入出力ストリームを用意し、クライアントからの要求を待つ
      Socket socket = serverSocket.accept();
      BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
      PrintStream writer = new PrintStream(socket.getOutputStream());
      // データを読み込み、そのまま返す
      while (true) {
        String line = reader.readLine();
        if (line == null)
          break;
        writer.println(line);
      }
      writer.close();
      reader.close();
      socket.close();
      serverSocket.close();
    } catch (SocketException e) { System.err.println("Socket
      error");   System.exit(-1);
    } catch (IOException e) {
      System.err.println("IO error");   System.exit(-1);
    }
  }
}
```

NWプログラミング

```
/*
 * SimpleServer.java: 受け取ったメッセージをそのまま返すサーバ
 */
import java.io.*;
import java.net.*;

public class SimpleServer {
    public static final int PORT = 5555;              // ポート番号

    public static void main (String args[]) {
        try {
            // ソケットを生成
            ServerSocket serverSocket = new ServerSocket(PORT);

            // 入出力ストリームを用意し，クライアントからの要求を待つ
            Socket socket = serverSocket.accept();
            BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            PrintStream writer = new PrintStream(socket.getOutputStream());
            // データを読み込み，そのまま返す
            while (true) {
                String line = reader.readLine();
                if (line == null) break;
                writer.println(line);
            }
            writer.close();
            reader.close();
            socket.close();
            serverSocket.close();
        } catch (SocketException e) {
            System.err.println("Socket error");   System.exit(-1);
        } catch (IOException e) {
            System.err.println("IO error");   System.exit(-1);
}}}
```

ここでソケットを生成して入出力のストリームを用意してやる.

ソケットに対して読み書きしやすいように、BufferReaderとかPrintStreamでラッピングしている.

NWプログラミング

```
/*
 * SimpleServer.java: 受け取ったメッセージをそのまま返すサーバ
 */
import java.io.*;
import java.net.*;

public class SimpleServer {
    public static final int PORT = 5555;                    // ポート番号

    public static void main (String args[]) {
        try {
            // ソケットを生成
            ServerSocket serverSocket = new ServerSocket(PORT);

            // 入出力ストリームを用意し，クライアントからの要求を待つ
            Socket socket = serverSocket.accept();
            BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            PrintStream writer = new PrintStream(socket.getOutputStream());
            // データを読み込み，そのまま返す
            while (true) {
                String line = reader.readLine();
                if (line == null) break;
                writer.println(line);
            }
            writer.close();
            reader.close();
            socket.close();
            serverSocket.close();
        } catch (SocketException e) {
            System.err.println("Socket error");   System.exit(-1);
        } catch (IOException e) {
            System.err.println("IO error");   System.exit(-1);
}}}
```

サーバはどこからアクセスが来るか
わからないので、5555のポートを
開けてずっと待っている

赤はソケットの設定で、皆さんはココを書き換えて
自分のサーバプログラムを作る

NWプログラミング

- Javaでの実装例（クライアント）

```
/*
 * SimpleClient.java: 送ったメッセージをそのまま受信するクライアント
 */
import java.io.*;
import java.net.*;

public class SimpleClient {
    public static final int PORT = 5555;          // ポート番号

    public static void main (String args[]) {

        try {                                                  // ソケットを作成
            Socket socket = new Socket(args[0]、PORT);
            BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            DataOutputStream writer = new DataOutputStream(socket.getOutputStream());

            // サーバにメッセージを送る
            writer.writeBytes("Hello\nWorld!\n");

            // サーバからメッセージを受け取る
            String line;
            while ((line = reader.readLine()) != null)
              System.out.println("Server: " + line);
            writer.close();
            reader.close();
            socket.close();
        } catch (UnknownHostException e) { System.err.println("Host
            not found");   System.exit(-1);
        } catch (SocketException e) { System.err.println("Socket
            error");   System.exit(-1);
        } catch (IOException e) {
            System.err.println("IO error");   System.exit(-1);
        }
    }
}
```

NWプログラミング

```
/*
 * SimpleClient.java: 送ったメッセージをそのまま受信するクライアント
 */
import java.io.*;
import java.net.*;

public class SimpleClient {
    public static final int PORT = 5555;                    // ポート番号

    public static void main (String args[]) {
        try {
            // ソケットを作成
            Socket socket = new Socket(args[0], PORT);
            BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            DataOutputStream writer = new DataOutputStream(socket.getOutputStream());

            // サーバにメッセージを送る
            writer.writeBytes("Hello\nWorld!\n");

            // サーバからメッセージを受け取る
            String line;
            while ((line = reader.readLine()) != null) System.out.println("Server: " + line);
            writer.close();
            reader.close();
            socket.close();
        } catch (UnknownHostException e) {
            System.err.println("Host not found");   System.exit(-1);
        } catch (SocketException e) {
            System.err.println("Socket error");   System.exit(-1);
        } catch (IOException e) {
            System.err.println("IO error");   System.exit(-1);
}}}
```

5555のポートを指定

赤はソケットの設定で、

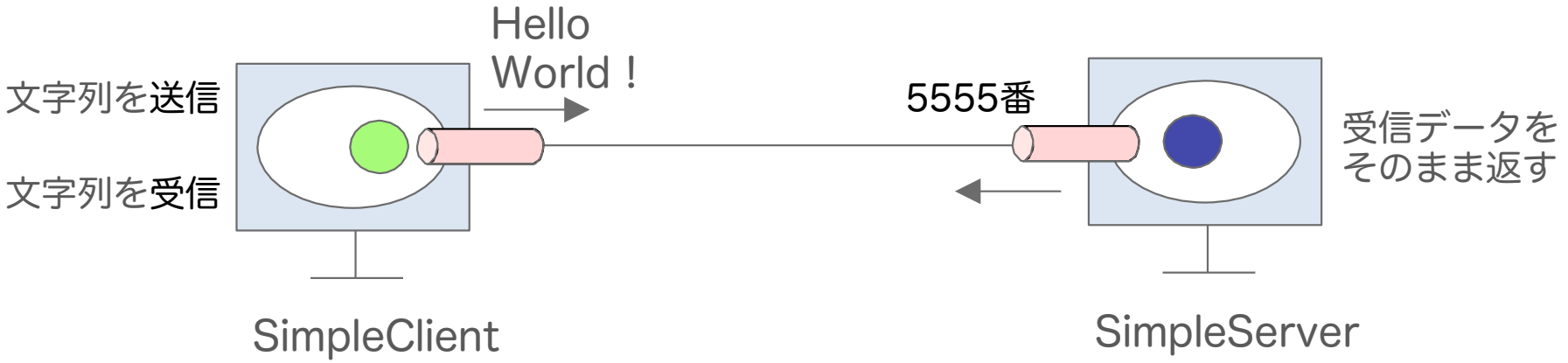
ココを書き換えて

サーバプログラムを作る

サンプルプログラムの実行結果

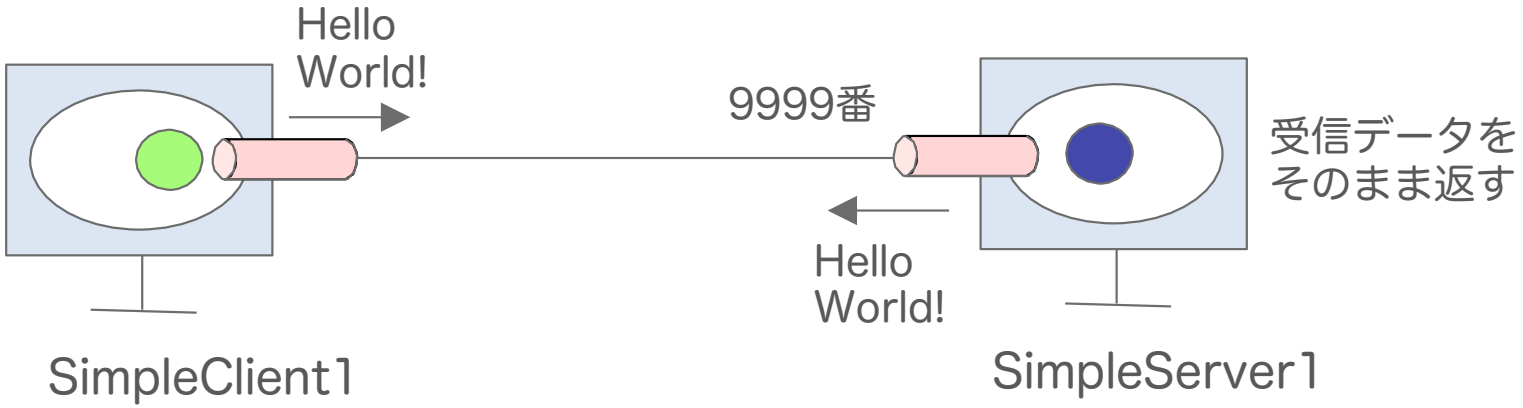
```
% javac  SimpleClient.java
%



% java SimpleClient localhost
Server: Hello
Server: World!
Ctrl-c
```

```
% javac  SimpleServer.java
%

% java SimpleServer
（受信待ち）



%
```



演習２課題イ①
SimpleServer1，SimpleClient1

- 下図のような動きをするSimpleServer1プログラムと，SimpleClient1プログラムを作成する．



- ポート番号をコマンドラインで指定

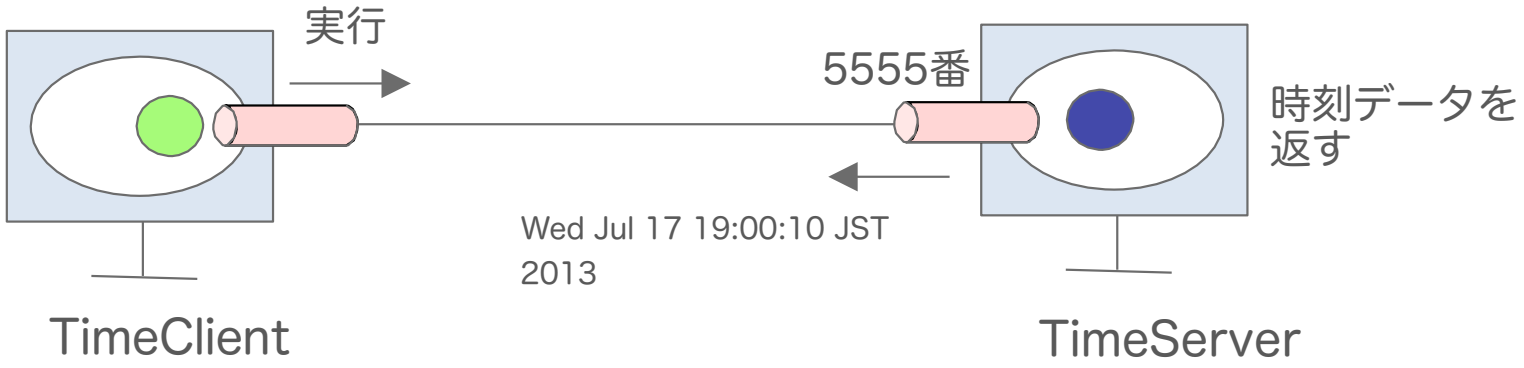
```
% java SimpleClient1 localhost 9999
Server: Hello
Server: World!
Ctrl-c
%
```

- ポート番号をコマンドラインで指定

```
% java SimpleServer1 9999
```

演習２課題イ②
TimeServer，TimeClient

- 下図のような動きをするTimeServerプログラムと，TimeClientプログラムを作成する.



- ポート番号をコマンドラインで指定

```
% java TimeClient localhost 5555
TimeServer: Wed Jul 17 19:00:10 JST 2013
%
```

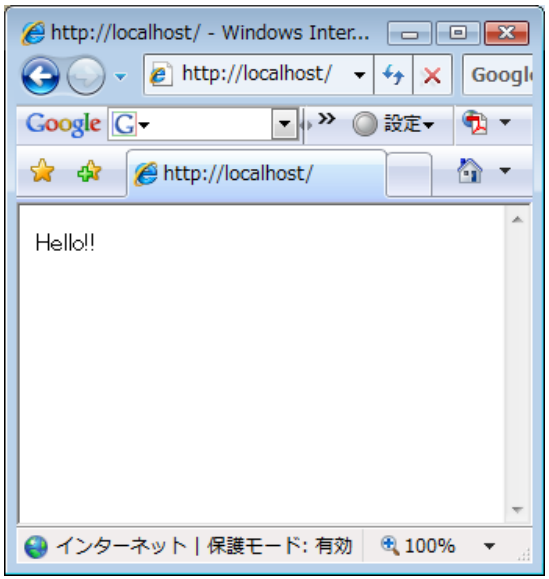
- ポート番号をコマンドラインで指定
- Ctrl−cで終了

```
% java TimeServer 5555
Ctrl-c
%
```

演習２課題イ③PhttpServer

- 下図のような動きを実現させるPhttpServerプログラムを作成する.

Webブラウザ



PhttpServer

- ブラウザからのアクセスを受け付ける
- 改行を2回読み飛ばすことでプロトコルを模擬
- 文字列を返す

```
% java PhttpServer 80
```

演習２課題イ④PhttpClient

- 下図のような動きを実現させるPhttpClientプログラムを作成する.

- Webサーバーにリクエストを送る
- デフォルトページを受信する

```
% java PhttpClient www.google.co.jp 80
HTTP/1.1 200 OK
Date: Wed,  17 Jul 2013 19:00:00 GMT
Expires: -1
Cache-Control: private,  max-age=0
Content-Type: text/html; charset=ISO-8859-1
…….


%
```

演習２課題イ
⑤OriginalServer，OriginalClient

- 各自で自由なサービスを実現させる，OriginalServerプログラムと，OriginalClientプログラムを作成する.

＜例＞

✳簡易FTPサーバとクライアント

✳チャットシステム

✳コーヒーポットプロトコル

✳遠隔地にあるコーヒーポットの制御を行う

✳RFCで標準化（RFC2324）