Κατανεμημένα Συστήματα

3ο Φυλλάδιο εργαστηρίου (16/03/2017)

Java Sockets – Server

Η πιο διαδεδομένη αρχιτεκτονική ανάπτυξης δικτυακών εφαρμογών είναι η αρχιτεκτονική του πελάτη - εξυπηρετητή (client - server). Ο εξυπηρετητής είναι μια διεργασία, η οποία εκτελείται σε έναν υπολογιστή και αναμένει να συνδεθεί σε αυτήν κάποιο πρόγραμμα - πελάτης, για να του παράσχει υπηρεσίες.

Η Java μας δίνει την δυνατότητα της ανταλλαγής δεδομένων μεταξύ προγραμμάτων που τρέχουν σε διαφορετικούς υπολογιστές. Αυτό επιτυγχάνεται με την δημιουργία sockets μεταξύ των δύο προγραμμάτων σε μια αρχιτεκτονική πελάτη-εξυπηρετητή.

Ως Socket ορίζεται το ένα άκρο, από έναν επικοινωνιακό δίαυλο διπλής κατεύθυνσης, μεταξύ δυο προγραμμάτων που εκτελούνται στο δίκτυο. Περιλαμβάνει το πρωτόκολλο, την διεύθυνση και τον αριθμό θύρας του άκρου.

Κάθε πρόγραμμα διαβάζει από και γράφει σε ένα socket, με χρήση Input και OutputStreams. Έτσι η διαδικασία αυτή δε διαφέρει σε τίποτα με την εγγραφή και ανάγνωση αρχείων. Μπορούμε να δημιουργήσουμε δύο είδη sockets ανάλογα με το πρωτόκολλο μεταφοράς που θα χρησιμοποιήσουμε :

- Στην πρώτη περίπτωση βασίζεται στο TCP (Transmission Control Protocol) και είναι μια υπηρεσία προσανατολισμένη στην σύνδεση (connection-oriented service). Μπορούμε να το θεωρήσουμε ανάλογο της τηλεφωνικής υπηρεσίας, στην οποία, μετά την εγκαθίδρυση μιας σύνδεσης μεταξύ δυο συνομιλητών, αυτή χρησιμοποιείται μέχρι το πέρας της συζητήσεως τους.

- Στην δεύτερη περίπτωση βασίζεται στο UDP (User Datagram Protocol) και είναι μια υπηρεσία χωρίς σύνδεση (connectionless service). Το ανάλογο, σε αυτήν την περίπτωση, είναι το ταχυδρομείο: μπορούμε να στείλουμε πολλά πακέτα στον ίδιο παραλήπτη, αλλά δεν είναι σίγουρο ότι όλα θα ακολουθήσουν την ίδια διαδρομή (σύνδεση) για να φτάσουν στον προορισμό τους.

Μια ακόμη σημαντική διαφορά, μεταξύ των παραπάνω δύο ειδών, είναι ότι τα TCP sockets εξασφαλίζουν μια αξιόπιστη μεταφορά της πληροφορίας, ότι αποστέλλεται από το ένα άκρο είναι σίγουρο ότι θα φτάσει στο άλλο. Στο UDP socket όμως δεν συμβαίνει αυτό. Είναι στην ευθύνη του αποστολέα να ελέγξει ότι αυτό που έστειλε, το έλαβε τελικά ο παραλήπτης και δεν χάθηκε στον δρόμο. Επιπλέον, η σύνδεση με TCP socket απαιτεί την ανταλλαγή τριών "πακέτων χειραψίας" (handshake packets) και είναι πιο χρονοβόρα στην αρχικοποίηση της από την αντίστοιχη με UDP datagrams. Οι προηγούμενες δύο διαφορές καθορίζουν τελικά και την χρήση των δύο αυτών ειδών.

Στην ορολογία της Java, ο όρος *Socket* χρησιμοποιείται για τα TCP sockets στην ονοματολογία των κλάσεων και των μεθόδων, ενώ για την δήλωση των UDP sockets, χρησιμοποιείται ο όρος *Datagram*.

Ο εξυπηρετητής (Server)

Για να δημιουργήσουμε έναν εξυπηρετητή (την εφαρμογή που περιμένει εισερχόμενες συνδέσεις) θα κάνουμε χρήση της κλάσης ***java.net.ServerSocket***. Σαν παράμετρο στον κατασκευαστή της κλάσης μπορούμε να περάσουμε την πόρτα στην οποία θα περιμένει ο εξυπηρετητής αίτημα για σύνδεση. Τέλος όταν δημιουργήσουμε το αντικείμενο εκτελούμε την μέθοδο **accept()** η οποία «μπλοκάρει» την εκτέλεση του προγράμματος μέχρι να έρθει κάποιο αίτημα. Η μέθοδος **accept()** επιστρέφει αναφορά σε αντικείμενο Socket, την οποία θα χρησιμοποιήσουμε στην συνέχεια για να ανταλλάξουμε μηνύματα με την άλλη πλευρά της σύνδεσης.

Ανταλλαγή μηνυμάτων

Στην συνέχεια για να ανταλλάξουμε μηνύματα μεταξύ των δύο προγραμμάτων θα πρέπει να πάρουμε το *InputStream* και *OutpuStream* της σύνδεσης με τις μεθόδους `InputStream` **`getInputStream`**`()` και `OutputStream` **`getOutputStream`**`()` για να για να λάβουμε και να στείλουμε αντίστοιχα. Οπότε τώρα χρησιμοποιώντας τις μεθόδους που ορίζονται στις κλάσεις *InputStream* και *OutpuStream* μπορούμε να ανταλλάξουμε δεδομένα. Τέλος για να κλείσουμε την σύνδεση τρέχουμε την μέθοδο `void` **`close`**`()` της κλάσης *java.net.Socket.*

Χρήσιμες Πληροφορίες

Επιπλέον στην κλάση *java.net.Socket* ορίζονται διάφορες μέθοδοι που μας δίνουν χρήσιμες πληροφορίες για την σύνδεση. Κάποιες από αυτές είναι:

```
InetAddress getLocalAddress()
InetAddress getInetAddress()
```

οι οποίες επιστρέφουν την τοπική και την απομακρυσμένη διεύθυνση της σύνδεσης.

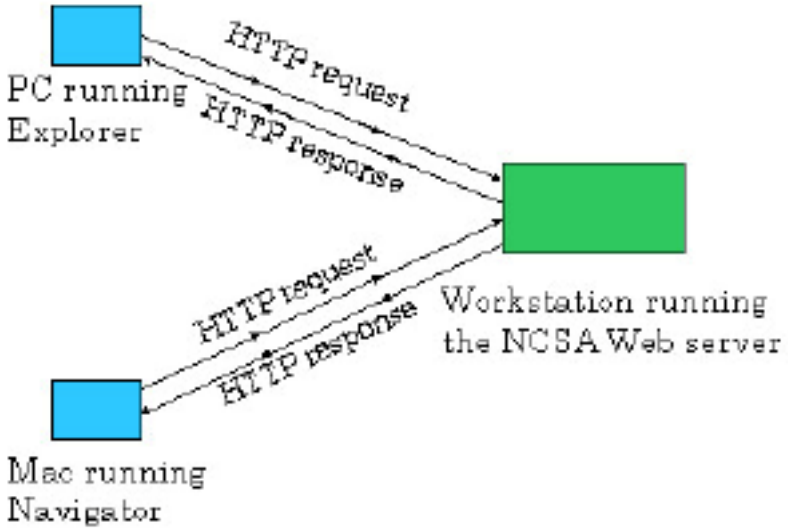
```
int getLocalPort()
int getPort()
```

οι οποίες επιστρέφουν την τοπική και την απομακρυσμένη πόρτα της σύνδεσης.

Το πρωτόκολλο HTTP

Το Πρωτόκολλο μεταφοράς Υπερκειμένου HTTP (HyperText Transfer Protocol) ανήκει στο στρώμα εφαρμογών του Διαδικτύου και ακολουθεί την αρχιτεκτονική πελάτη - εξυπηρετητή. Οι εφαρμογές πελάτη και εξυπηρετητή εκτελούνται σε διαφορετικά μηχανήματα επικοινωνώντας μεταξύ τους ανταλλάσσοντας HTTP μηνύματα. Συγκεκριμένα το HTTP ορίζει τη δομή των μηνυμάτων αυτών καθώς και τον τρόπο ανταλλαγής τους ανάμεσα στον πελάτη και στον εξυπηρετητή. Πριν περιγράψουμε αναλυτικά το πρωτόκολλο HTTP πρέπει να αναφερθούμε σε κάποια βασική ορολογία του ιστού. Μία ιστοσελίδα (Web page) αποτελείται από αντικείμενα. Με τον όρο αντικείμενο (object) εννοούμε ένα απλό αρχείο, όπως ένα αρχείο HTML, ένα αρχείο εικόνας ή ένα αρχείο βίντεο, το οποίο μπορεί να προσπελαστεί μέσω ενός URL. Οι περισσότερες ιστοσελίδες αποτελούνται από ένα βασικό αρχείο HTML και διάφορα σχετικά αντικείμενα. Το βασικό αρχείο HTML αναφέρεται στα άλλα αντικείμενα της σελίδας μέσω των URL των αντικειμένων. Κάθε URL αποτελείται από δύο τμήματα: το όνομα του υπολογιστή – host στον οποίον είναι αποθηκευμένο το αρχείο και το όνομα του μονοπατιού (path) του

αντικειμένου. Π.χ. το URL www.icsd.aegean.gr/index.htm έχει ως όνομα host το www.icsd.aegean.gr και ως όνομα μονοπατιού το index.htm. Ο φυλλομετρητής απεικονίζει στον χρήστη τη ζητούμενη ιστοσελίδα και παρέχει πληθώρα χαρακτηριστικών πλοήγησης και παραμετροποίησης. Επίσης, στους φυλλομετρητές υλοποιείται και η πλευρά του πελάτη του πρωτοκόλλου HTTP. Ένας εξυπηρετητής ιστού (Web server) αποθηκεύει τα αντικείμενα της ιστοσελίδας, το καθένα από τα οποία έχει ως διεύθυνση ένα URL. Στους εξυπηρετητές ιστού υλοποιείται και η πλευρά του εξυπηρετητή του πρωτοκόλλου HTTP. Το HTTP ορίζει τον τρόπο με τον οποίο οι πελάτες του ιστού αιτούνται (request) ιστοσελίδες από τους εξυπηρετητές του ιστού και πως οι εξυπηρετητές μεταφέρουν τις ιστοσελίδες στους πελάτες (response). Η βασική ιδέα του πρωτοκόλλου αυτού φαίνεται στο παρακάτω σχήμα.



Όταν ο χρήστης ζητά μία ιστοσελίδα, ο φυλλομετρητής στέλνει ένα μήνυμα HTTP αίτησης (HTTP request), για τα διάφορα αντικείμενα της σελίδας, στον εξυπηρετητή. Ο εξυπηρετητής όταν λάβει το μήνυμα αυτό ανταποκρίνεται με μηνύματα HTTP απόκρισης (HTTP response) στα οποία περιέχονται τα αιτούμενα αντικείμενα. Το HTTP πρωτόκολλο χρησιμοποιεί το TCP ως πρωτόκολλο μεταφοράς. Αφού ο πελάτης εγκαθιδρύσει μία σύνδεση TCP με τον εξυπηρετητή αρχίζει την αποστολή μηνυμάτων – αιτήσεων προς αυτόν και τη λήψη μηνυμάτων – αποκρίσεων από αυτόν. Λόγω της χρήσης του TCP το HTTP δεν χρειάζεται να ασχοληθεί καθόλου με τη μεταφορά των δεδομένων. Το μόνο που πρέπει να κάνει είναι να στείλει τις αιτήσεις μέσω της TCP σύνδεσης και να περιμένει τις αποκρίσεις. Το TCP εγγυάται την αξιόπιστη μεταφορά των δεδομένων καθώς και τον έλεγχο της συμφόρησης. Οι εξυπηρετητές του HTTP δεν κρατάνε καθόλου στοιχεία για την κατάσταση του πελάτη. Επομένως, αν ένας πελάτης στείλει μία αίτηση για ένα αρχείο δύο φορές, ο εξυπηρετητής θα του στείλει το αρχείο αυτό δύο φορές. Τα πρωτόκολλα που δεν κρατάνε καθόλου πληροφορία για την κατάσταση του πελάτη ονομάζονται stateless.

Μορφή των HTTP μηνυμάτων

Το HTTP ορίζει μόνο δύο τύπους μηνυμάτων: τις HTTP αιτήσεις (requests) και τις HTTP αποκρίσεις (responses).

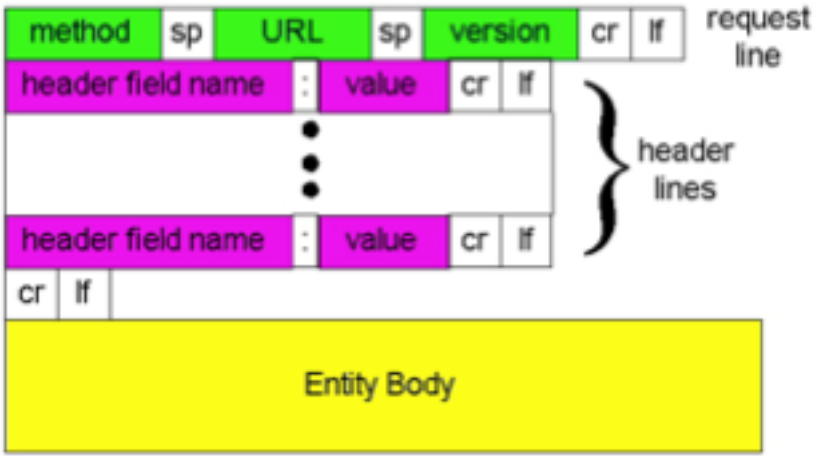
Ακολουθεί ένα παράδειγμα ενός μηνύματος HTTP αίτησης.

```
GET /lessons/index.htm HTTP/1.1
Host: www.icsd.aegean.gr
Connection: close
Accept: text/html, image/jpg
User-Agent: Mozilla/5.0
Accept-Language: el-GR,el;q=0.8

(extra carriage return,line feed)
```

Όπως φαίνεται και από το παραπάνω παράδειγμα οι HTTP αιτήσεις γράφονται με χαρακτήρες ASCII. Οι περισσότερες HTTP αιτήσεις αποτελούνται από 6 γραμμές κειμένου ακολουθούμενες από μία κενή γραμμή. Η πρώτη γραμμή κειμένου ονομάζεται γραμμή αίτησης (request line), ενώ οι επόμενες γραμμές επικεφαλίδας (header lines). Η γραμμή αίτησης περιέχει τρία πεδία: το πεδίο μεθόδου, το πεδίο URL και το πεδίο HTTP έκδοσης. Το πεδίο μεθόδου μπορεί να έχει μία από τις ακόλουθες τιμές: GET, POST και HEAD. Η πιο συνηθισμένη μέθοδος στις HTTP αιτήσεις είναι η GET, με την οποία ζητείται από τον εξυπηρετητή η αποστολή του αρχείου που εμφανίζεται στο πεδίο URL. Στο URL δεν είναι απαραίτητο να υπάρχει και το όνομα του host, αφού ήδη έχει εγκατασταθεί μία σύνδεση με τον host αυτό (δηλαδή τον HTTP εξυπηρετητή). Τέλος, στο πεδίο έκδοσης HTTP περιγράφεται η έκδοση του HTTP που χρησιμοποιεί ο πελάτης. Στο παραπάνω παράδειγμα χρησιμοποιείται το HTTP/1.1. Η γραμμή Connection:close λέει στον εξυπηρετητή να τερματιστεί η σύνδεση μετά την αποστολή του αρχείου, δηλαδή να μη γίνει χρήση μόνιμης σύνδεσης. Η γραμμή που αρχίζει με User-Agent: δηλώνει τον τύπο του φυλλομετρητή του χρήστη. Στο παραπάνω παράδειγμα δηλώνεται ότι ο χρήστης χρησιμοποιεί τον φυλλομετρητή Mozilla/5.0. Η γραμμή αυτή χρησιμοποιείται από τον εξυπηρετητή για την αποστολή διαφορετικών αντικειμένων ανάλογα με τον φυλλομετρητή του χρήστη (τα οποία όμως έχουν το ίδιο URL). Η Accept: γραμμή δηλώνει τον τύπο των αντικειμένων που υποστηρίζει ο φυλλομετρητής. Στο παραπάνω παράδειγμα δηλώνεται ότι ο φυλλομετρητής υποστηρίζει αρχεία HTML και αρχεία εικόνων τύπου JPEG. Τέλος, η Accept-Language: γραμμή δηλώνει την γλώσσα έκδοσης του αντικειμένου που επιθυμεί να λάβει ο φυλλομετρητής. Αν ο εξυπηρετητής δεν έχει έκδοση του αντικειμένου για την γλώσσα αυτή τότε στέλνει την προκαθορισμένη έκδοση του αντικειμένου. Στο παραπάνω παράδειγμα αν ο εξυπηρετητής διαθέτει μία ελληνική έκδοση του αρχείου lessons/index.htm τότε πρέπει να τη στείλει, αλλιώς να στείλει την προκαθορισμένη έκδοση του αρχείου.

Η δομή του μηνύματος HTTP αίτησης φαίνεται στο παρακάτω σχήμα.



Από το παραπάνω σχήμα βλέπουμε ότι η γενική μορφή του μηνύματος ακολουθεί τη δομή του παραπάνω παραδείγματος. Το πεδίο Entity Body δεν χρησιμοποιείται για τη μέθοδο GET, χρησιμοποιείται όμως όταν γίνεται χρήση της μεθόδου POST. Ένα παράδειγμα χρήσης της μεθόδου POST είναι η αποστολή των δεδομένων που συμπλήρωσε ο χρήστης σε διάφορες φόρμες μίας ιστοσελίδας. Παρόμοια χρήση του πεδίου Entity Body γίνεται και στην περίπτωση χρήσης της μεθόδου HEAD. Η μέθοδος αυτή είναι χρήσιμη στον εντοπισμό σφαλμάτων (debugging) από τους developers των εξυπηρετητών Ιστού: όταν ο εξυπηρετητής λάβει αίτηση με χρήση της μεθόδου HEAD αποκρίνεται με ένα HTTP μήνυμα στο οποίο δεν περιλαμβάνεται το αρχείο.

Παρακάτω φαίνεται ένα παράδειγμα μηνύματος HTTP απόκρισης.

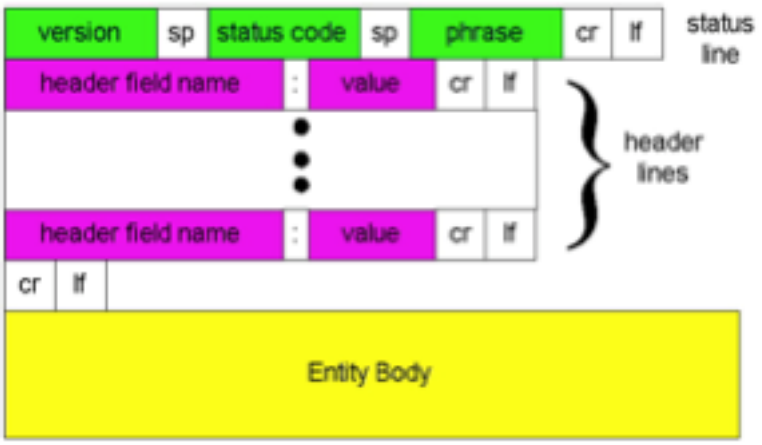
```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 13 Mar 2014 18:38:53 GMT
Server: Microsoft-IIS/6.0
Last-Modified: Mon, 22 Jul 2013 09:20:40 GMT
Content-Length:6821
Content-Type: text/html; charset=utf8

data data data data data …
```

Το HTTP μήνυμα απόκρισης αποτελείται από τρία μέρη: την γραμμή κατάστασης (status line), τις γραμμές επικεφαλίδας (header lines) και το τμήμα περιεχομένου (entity body). Το τμήμα περιεχομένου περιέχει τα δεδομένα του αιτούμενου αρχείου. Η γραμμή κατάστασης περιέχει τρία πεδία: το πεδίο HTTP έκδοσης, το πεδίο κωδικού κατάστασης και το πεδίο του αντίστοιχου μηνύματος κατάστασης. Στο παραπάνω παράδειγμα δηλώνεται ότι γίνεται χρήση του HTTP/1.1 και ότι δεν παρουσιάστηκε κάποιο σφάλμα (κωδικός κατάστασης 200 και αντίστοιχο μήνυμα κατάστασης OK). Η γραμμή Connection:close δηλώνει ότι μετά την αποστολή του μηνύματος αυτού η TCP σύνδεση θα τερματιστεί. Η Date: γραμμή δηλώνει την ημερομηνία και την ώρα κατά την οποία δημιουργήθηκε και στάλθηκε η απόκριση, η Server: γραμμή δηλώνει τον τύπο του εξυπηρετητή ιστού, η Last-Modified: δηλώνει την ημερομηνία και την ώρα κατά την οποία το αντικείμενο δημιουργήθηκε ή τροποποιήθηκε για τελευταία φορά, η γραμμή Content-Length: δηλώνει το μήκος των

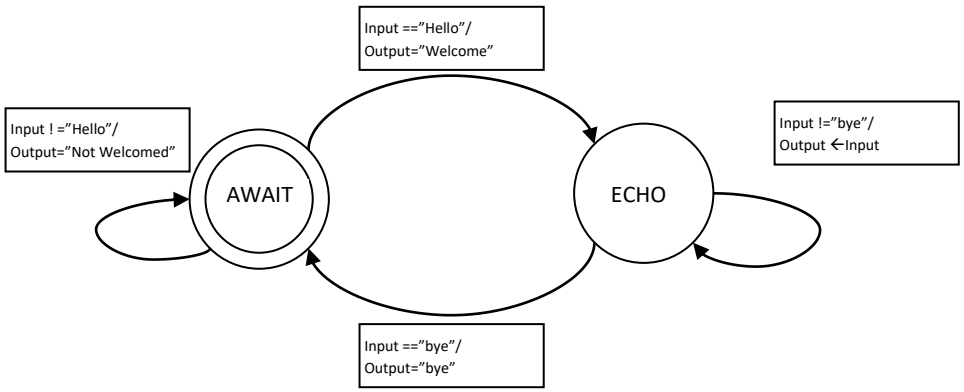
δεδομένων που αποστέλλονται (σε bytes), ενώ η γραμμή Content-Type: δηλώνει τον τύπο του αντικειμένου που περιέχεται στο τμήμα περιεχομένου του μηνύματος. Στο παραπάνω παράδειγμα δηλώνεται ότι χρησιμοποιείται ο Microsoft-IIS/6.0 εξυπηρετητής ιστού και ότι αποστέλλεται ένα αρχείο HTML μήκους 6821 bytes.

Η δομή των HTTP μηνυμάτων απόκρισης φαίνεται στο παρακάτω σχήμα.



1η Εργαστηριακή Άσκηση – Επικοινωνία Client-Server

Καλείστε να δημιουργήσετε έναν εξυπηρετητή που θα αναμένει αιτήματα από έναν πελάτη. Ο εξυπηρετητής θα βρίσκεται στην τοπική σας διεύθυνση και θα περιμένει συνδέσεις στην θύρα 5555. Θα πρέπει να υλοποιεί το παρακάτω πρωτόκολλο. Δεν απαιτείται ο εξυπηρετητής να δέχεται πολλές συνδέσεις ταυτόχρονα.



Ενδεικτική έξοδος της εφαρμογής μπορεί να είναι η ακόλουθη:

Accepting Connection...
Local Address /195.251.161.155 and port 5566
Connected from :/ 195.251.161.147 Port :24499
The client says: Hello
The server says: Welcome
The client says: hi
The server says: hi

The client says: bye
The server says: bye
Connection Closing...

Ενδεικτική Απάντηση

Server

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

public class EchoServer {

    public static void main(String[] args) {

      try{
        ServerSocket server = new ServerSocket(5555,50);

        while (true){
        System.out.println("Accepting Connection...");
        System.out.println("Local Address :"+server.getInetAddress()+" Port
:"+server.getLocalPort());
        Socket sock = server.accept();

        BufferedReader instream = new BufferedReader (new InputStreamReader
(sock.getInputStream()));
        BufferedWriter outstream = new BufferedWriter(new
OutputStreamWriter(sock.getOutputStream()));

        String strin = instream.readLine();
         System.out.println("The client says : " + strin);

        if (strin.equals("Hello")){ //following the protocol
                outstream.write("Welcome"+"\n");
                outstream.flush();

                do{
                    strin = instream.readLine();
                     System.out.println("The client says : " + strin);

                    outstream.write(strin+"\n");
                    outstream.flush();

                }while(!strin.equals("bye"));  //bye = terminate the conversation
                }
        else {  //not following the protocol
            outstream.write("Not welcomed..."+"\n");
            outstream.flush();

        }
        instream.close();
        outstream.close();
        sock.close();
        System.out.println("Connection Closing...");

        }
       }
      catch (Exception ex){
            System.out.println("Error during I/O");
            ex.getMessage();
            ex.printStackTrace();
        }    } }
```

2η Εργαστηριακή Άσκηση – Δημιουργία Web Client

Δημιουργήστε ένα πρόγραμμα (client) που θα συνδέεται στην ιστοσελίδα του τμήματος και θα εμφανίζει το περιεχόμενο της σε plaintext μορφή. Τι αίτημα πρέπει να στείλει ο «πελάτης» σας και σε ποια μορφή;

Ενδεικτική Απάντηση

Το αίτημα που θα πρέπει να σταλεί από το πρόγραμμα μας προς τον web εξυπηρετητή του τμήματος θα πρέπει να είναι της μορφή : "GET /icsd/ HTTP/1.1\nHost: www.icsd.aegean.gr\n\n".

Socket Web Client

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

public class WebClient {
    public static void main(String[] args)  {
      try{

         String address = "www.icsd.aegean.gr";
        int port = 80;
        Socket sock = new Socket(address, port);

        BufferedReader instream = new BufferedReader(new InputStreamReader
(sock.getInputStream()));
        BufferedWriter outstream = new BufferedWriter(new
OutputStreamWriter(sock.getOutputStream()));

        String request = "GET /icsd/ HTTP/1.1\nHost: www.icsd.aegean.gr\n\n";
        System.out.println("Sending Messages to the Server... :"+request);
        System.out.println("Connecting to "+ sock.getInetAddress()+ " and port
"+sock.getPort());
        System.out.println("Local Address :"+sock.getLocalAddress()+" Port
:"+sock.getLocalPort());
           String response;
           outstream.write(request);
           outstream.flush();
           System.out.println("The server says: ");
           while ( (response = instream.readLine()) != null)
              System.out.println(response);
        instream.close();
        outstream.close();
        sock.close();
        System.out.println("Connection Closing...");
        }
      catch (IOException ex){
          System.out.println("Error during I/O");
          ex.getMessage();
          ex.printStackTrace();
        }
    }
}
```

3η Εργαστηριακή Άσκηση – Δημιουργία Web Server

Δημιουργήστε έναν εξυπηρετητή που θα περιμένει αιτήματα στην θύρα 8080 και θα προσομοιώνει την λειτουργία ενός WEB εξυπηρετητή. Δηλαδή θα δέχεται αιτήματα (HTTP Request) και θα απαντά στον πελάτη με μία ολοκληρωμένη απόκριση (HTTP Response).

α) Συνδεθείτε στον εξυπηρετητή μέσω ενός φυλλομετρητή στην διεύθυνση *http:\\localhost:8080*.
β) Εκτελέστε την προηγούμενη άσκηση (Web Client) και συνδεθείτε σε αυτόν τον εξυπηρετητή που υλοποιήσατε.

Ενδεικτική Απάντηση

Socket Web Server

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

public class WebServer {

	public static void main(String args[]) {

		ServerSocket server;

		try {

			server = new ServerSocket(8080);

			while (true){

				Socket sock = server.accept();
				System.out.println("Connection accepted");
					BufferedReader input = new BufferedReader(new
				InputStreamReader(sock.getInputStream()));
					BufferedWriter output = new BufferedWriter(new
				OutputStreamWriter (sock.getOutputStream()));

				String str = null;
				do{
					str = input.readLine();
				}
				while (!str.equals(""));  //read until black line


				output.write("HTTP/1.0 200 OK"+"\n");
				output.write("Content-Type: text/html"+"\n");
				output.write("Date : "+ new Date()+"\n");
				output.write("Server: MyApache Java based"+"\n");
				output.write(""+"\n"); //empty line - end of headers
				output.write("<title>My First Page</title>"+"\n");
				output.write("<h1>Welcome to Distributed System
Official page</h1>"+"\n");  // HTML page
				output.flush();
				sock.close();
				System.out.println("Connection closed");

			}
```

```
                }
            catch(IOException ex){

                    ex.printStackTrace();
            }
        }
}
```

Ανάπτυξη Client – Server εφαρμογής με UDP Datagrams

Οι έννοιες του πελάτη και του εξυπηρετητή δεν είναι και τόσο διακριτές όταν υλοποιούμε με χρήση UDP datagrams. Σε αυτή την περίπτωση δεν δίνεται έμφαση στην σύνδεση (δεν υπάρχει άλλωστε με την τυπική έννοια, όπως στο TCP), αλλά στη σωστή δημιουργία του πακέτου που πρόκειται να σταλεί, έτσι ώστε να περιέχει όλες τις απαραίτητες πληροφορίες, μαζί με τα δεδομένα που θέλουμε να σταλούν.

Θα χρησιμοποιήσουμε τα αντικείμενα δύο κλάσεων : την κλάση **DatagramSocket**, που υλοποιεί την επικοινωνία με UDP datagrams και την κλάση **DatagramPacket**, που μας επιτρέπει να διαμορφώσουμε τα προς αποστολή πακέτα.

Παράδειγμα Εφαρμογής Echo

Πελάτης (client)

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

public class EchoClientUDP
{
private static InetAddress host;
private static final int PORT = 1234;
private static DatagramSocket dgramSocket;
private static DatagramPacket inPacket, outPacket;
private static byte[] buffer;

public static void main(String[] args)
    {
    try
        {

       // Καθορίζουμε τη διεύθυνση του Host που τρέχει τον εξυπηρετητή
        host = InetAddress.getByName("localhost");
        }
    catch(UnknownHostException e)
        {
        System.out.println("Host ID not found!");
        System.exit(1);
        }

    try
        {

// Δημιουργούμε το datagram socket, μέσω του οποίου θα στείλουμε το πακέτο
// με την αίτηση και θα λάβουμε την απάντηση.
// Παρατηρούμε ότι δεν χρειάζεται να συνδέσουμε το datagram socket με ροές
// εισόδου/εξόδου, αφού δεν έχουμε I/O σε κάποιο κανάλι, αλλά αποστολή και
// λήψη μεμονωμένων πακέτων
```

```
        dgramSocket = new DatagramSocket();

        Scanner input = new Scanner(System.in);
        String message="", response="";

         do
        {
        System.out.print("Enter message: ");
        message = input.next();

         if (!message.equals("bye"))
             {

// Με την DatagramPacket κατασκευάζουμε το πακέτο. Ο constructor της κλάσης
// δέχεται 4 ορίσματα : έναν buffer, ο οποίος αποτελεί τα δεδομένα σε μορφή
// πίνακα από bytes, το μέγεθος του προηγούμενου πίνακα, την InetAddress
// διεύθυνση προορισμού, καθώς και τον αριθμό θύρας του προορισμού.

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


// Αποστολή του πακέτου με την μέθοδο send του datagram socket
             dgramSocket.send(outPacket);

             buffer = new byte[256];
             inPacket = new DatagramPacket(buffer, buffer.length);

// Αναμονή για λήψη της απάντησης με τη μέθοδο receive του datagram socket

// Η receive μέθοδος μπλοκάρει το νήμα από το οποίο έχει κληθεί,  μέχρι να
// λάβει κάποιο datagram  πακέτο στο port που έχει ανοίξει το datagram
// socket. Χρειάζεται προσοχή, καθώς το UDP πρωτόκολλο δεν είναι αξιόπιστο
// και το πακέτο με την απάντηση μπορεί να χαθεί (συνίσταται η
// χρησιμοποίηση timer για να ξεμπλοκάρει το νήμα και  να τερματιστεί ομαλά
// η εφαρμογή).

             dgramSocket.receive(inPacket);

// Με την μέθοδο getData λαμβάνουμε τον πίνακα από bytes με την πληροφορία
// που θέλουμε.
             response = new String(inPacket.getData(), 0, inPacket.getLength());
             System.out.println("\nSERVER> " + response);
             }
         else
             break;
        }
         while (true);
        }

         catch(IOException e)
             {
             e.printStackTrace();
             }

          finally
               {
                System.out.println("\n* Closing connection... *");

// Απελευθερώνουμε τους πόρους που έχουμε δεσμεύσει, δηλαδή το datagram
// socket.
               dgramSocket.close();
               }
        }
}
```

Εξυπηρετητής (Server)

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

public class EchoServerUDP
{
private static DatagramSocket dgramSocket;
private static DatagramPacket inPacket, outPacket;
private static byte[] buffer;

public static void main(String[] args)
    {
    System.out.println("Opening connection...\n");
    try
        {

// Ο εξυπηρετητής δημιουργεί το datagram socket, για την λήψη των πακέτων.
// Ο constructor του αντικειμένου DatagramSocket δέχεται το port στο οποίο
// «ακούει» το socket, για νέες συνδέσεις

        dgramSocket = new DatagramSocket(1234);  //Port = 1234
        }
    catch(SocketException e)
        {
        System.out.println("Unable to use this port!");
        System.exit(1);
        }

    try
        {
       String messageIn, messageOut;
       int numMessages = 0;

       do
        {
       buffer = new byte[256];
       inPacket = new DatagramPacket(buffer, buffer.length);
       dgramSocket.receive(inPacket);

       InetAddress clientAddress = inPacket.getAddress();
       int clientPort = inPacket.getPort();
       messageIn = new String(inPacket.getData(), 0, inPacket.getLength());

       System.out.println("Message received.");
       numMessages++;
       messageOut = ("Message " + numMessages + ": " + messageIn);

        outPacket = new DatagramPacket(messageOut.getBytes(), messageOut.length(),
clientAddress, clientPort);
       dgramSocket.send(outPacket);

       }while (true);
     }

    catch(IOException e)
       {
       e.printStackTrace();
       }

    finally
        {
```

```
    System.out.println("\n* Closing connection... *");
    dgramSocket.close();
        }
    }
}
```