**Receiver**

Receiver sunucu olduğundan görevi **sender**’in bağlanacağı portu **sürekli olarak** dinlemektir. Bunun için port numarası yeterlidir.

**import** java.io.InputStream;  
**import** java.io.PrintWriter;  
**import** java.net.Socket;  
**import** java.util.Scanner;  
  
  
**public class** Receiver **extends** Root{  
 Socket **link** = **null**;  
 Scanner **input**;  
 PrintWriter **output**;

İlk olarak java’da soket programlamada kullanılan kütüphaneleri içe aktarıyoruz. Ardından scanner, socket ve printwriter komutlarımızı yazıyoruz. Bu komutları veri alma, soket oluşturma ve yazdırma için kullanacağız.

**public** Receiver(**int** PORT) {  
 **this**.openListenPort(PORT);  
 }  
  
  
kodunu da başta belirtilen sender’in bağlanacağı portu dinlemek için kullanıyoruz.

@Override  
 **public void** run() {  
 **while** (**true**) {   
 **try** {  
**link** = serverSocket.accept();  
**input** = **new** Scanner(**link**.getInputStream());**output** = **new** PrintWriter(**link**.getOutputStream(), **true**); Thread th = **new** ClientHandler(**link**,**input**,**output**);  
th.start();  
 } **catch** (IOException ex) {  
 System.err.println(**"Root.Receiver.startService()"**);  
 }  
 }  
   
 }

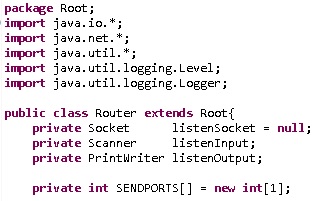
Yukarıdaki kodda while döngüsüne girdik ve link=serverSocket.accept() ile gelen istekleri kabul edeceğiz. getInputStream ile verileri okuyoruz. (router’dan gelecek olan). getOutput ile de sokete mesajı yazdırma işlemlerini gerçekleştiriyoruz. Ardından thread oluşturuyoruz ve threadi çalıştırıyoruz.

**private void** handleRouter() **throws** IOException {  
  
 String message;  
 **do** {  
 message = **this**.getMessage();  
 **this**.sendMessage(**"ACK"**+ message.substring(message.length() - 1));  
 System.out.println(**" - "** +message);  
 } **while** (!message.equals(**"\*\*\*CLOSE\*\*\*"**));  
  
}

getMessage ile paketten gelen bilgi mesajını alıyoruz. Ardından gelen mesajın bitip bitmediğini kontrol ediyor, eğer mesaj geldiyse “ACK” dönüyor. Gelen mesajı karakter sayısından 1 indis kısaltıp “ACK” mesajı ile beraber mesajı alıp almadığını bildiriyor.

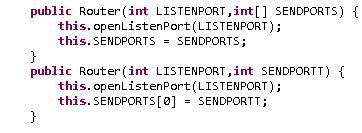
While döngüsünde de equals ile “CLOSE” gelirse bağlantıyı kapat diye şart kullanıyoruz.

**ROUTER**



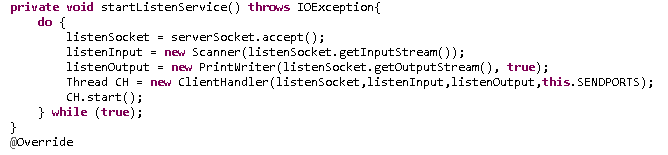
Şekil 1

1. Burada öncelikle gerekli olan kütüphaneleri import ederek başladık.
2. Router’ı Root’a extends ettik yani. Root sınıfındaki değişkenlere sahip olduk.
3. ListenSocket ile soketi dinledik. ListenInput (Scanner) ile gelen veriyi okuduk. ListenOutput (PrintWriter) ile Router’dan soketimize bilgi yazmak için kullandık.
4. Arından port numarası aldık (maksimum 2 adet)



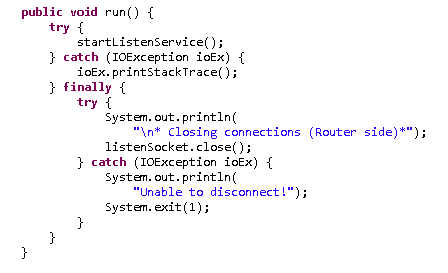
Şekil 2

1. Burada Constratcor yapı ile karışlaşmaktayız.
2. Bu iki yapıda tek portu dinlemekte.
3. Bunlardan biri iki porta gönderim sağlarken diğeri tek bir porta gönderim sağlamakta.



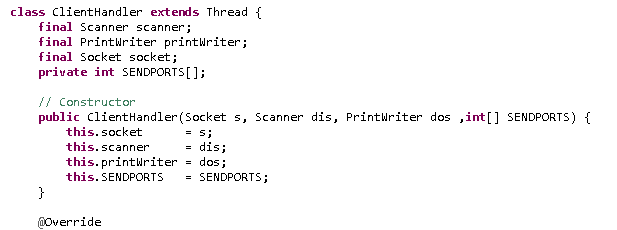
Şekil 3

1. Router startListenService çağırıldığınıda sürekli port dinlemeye başlıyor.
2. Gelen veriyi okumak içi Scanner komutunu kullanıyoruz.
3. Veri gönderimi yapmak için PrintWriter komutunu kullanıyoruz.
4. Thread CH =… kısmında Thread’ımızı oluşturuyoruz.
5. Ch.Start() kısmında ise Thread’ımızı başlatıyoruz.



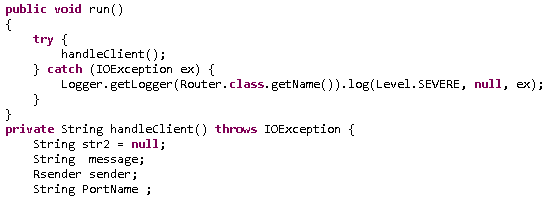
Şekil 4

1. İlk try bloğunda startListenService metoduna dönüp hataya bakıyor.
2. Catch kısmına hata varsa giriyor.
3. Finally bloğunun altında try bloğunda soketi kapatıyor.
4. Catch bloğunda ise Soket kapanmazsa hata ile karşılaşırsa uygulamayı kapatıyor.
5. En sonda ise tekrardan startListenService metoduna geri dönüyor.



Şekil 5

1. Gelen veriyi okumak içi Scanner komutunu kullanıyoruz.
2. Veri gönderimi yapmak için PrintWriter komutunu kullanıyoruz.
3. Buradaki Constructor yapı göderilecek port numaralarıyla gönderiliyor.



Şekil 6

1. Burada en başta handleClient adlı metoda gidiyor.



Şekil 7

1. Do-While döngüsü ile karşılaşıyoruz..
2. İlk Başta gelen veri alınıyor ve ekrana yazdırılıyor.
3. Random sayı oluşturmak için bir sınıf oluşturuluyor ve 0-99 arasında random sayı oluşturuluyor.
4. Oluşturulan random sayı paket kaybı amacıyla kullanılıyor.
5. Burada if else yapısı ile karşılaşıyoruz
6. Eğer random sayımız 19’dan küçükse if’e giriyor değil ise else’ye giriyor.
7. Ardından tekrardan if else yapısı ile karşılaşıyoruz burada port uzunluğuna göre bir koşul koşulmakta
8. Eğer port numarası 1’den farklı ise if’e, port numarası 1 ise else’ye giriyor.

* İlk if’te getroute'a gidiyor. port sayısının uzunluğuna göre getroute sayısı oluşturuyor.
* Örnek olarak 2 port numarası geldiyse ya 1 yada 0 oluyor. tek port numarası geldiyse sadece 0 oluşturuyor. kısaca iki porttan birini seçebilmek için random bir sayı oluşturuluyor.
* Ardından port adı alınıyor ve getPorName metoduna gidiyor
* Gönderim içi sender oluşturuyor.

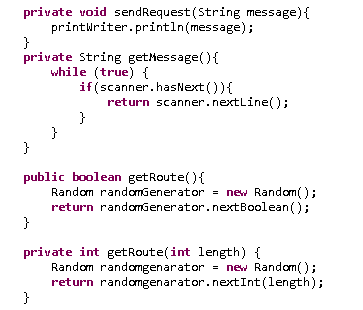
1. Else’de yani port uzunluğu 1 ise giriyor.

* Ardından port numarası gönderim sender oluşturuyor.
* Dinlenen porta mesaj yollanıyor ve mesaj yolladığı soketi kapatıyor.
* En sonda ise Java çöp toplayıcısını çalıştırıyor hafıza kaybı olmaması için.

1. Dışardaki else’ye girdiğinde ise

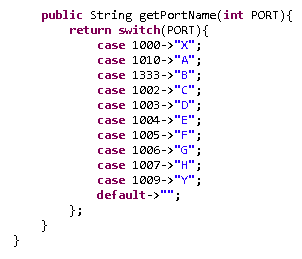
* Boş mesaj gönderiyor veri kaybı yapılsın diye.

1. En sonda ise gelen mesaj Close’a eşit olana kadar devam ediyor döngümüz.



Şekil 8

1. Burada en üstteki 2 metod ile alıcıdan mesaj alımı ve alıcıya cevap gönderiyoruz.
2. SendRequest (String message) ile mesaj yolluyoruz.
3. GetMessage ile de mesaj alıyoruz.
4. Ardından getRoute metodunda tekrardan random sayı oluşturuluyor (burada kodlar bağlantılı yukarıda bahsettiğim gibi aslında getroute’ye önceden geldik ama kodun akışına göre anlatım sağladığım için bu şekilde gidiyor.)



1. Burada ise Port numarasına göre port adını geri döndürüyoruz.

**SENDER**

package Root;

import java.io.\*;

import java.net.\*;

import java.util.\*;

public class Sender extends Thread {

private InetAddress host;

private final int PORT;

private Socket link = null;

Scanner input;

PrintWriter output;

public Sender(int port) {

this.PORT = port;

this.serviceStart(PORT);

}

@Override

public void run() {

try {

while (true) {

this.accessServer();

}

}

catch (IOException ioEx) {

ioEx.printStackTrace();

}

finally{

try {

//Socketi kapatıyor bilgilendirme mesajı veriyor

System.out.println("\n\* Closing connections (Sender side)\*");

output.println("\*\*\*CLOSE\*\*\*");

link.close();

}

catch (IOException ioEx) {

//Socketi kapatırken hata ile karşılaşırsa bilgi veriyor

System.out.println("Unable to disconnect!");

System.exit(1);

}

}

}

//Try catch bloguna ihtiyaç duymadan bu fonksiyonun throws ile hata atabileceğini bildiriyor

private void accessServer() throws IOException {

System.out.println("How many packets? ");

Scanner userEntry = new Scanner(System.in);

String message, str2, response;

int number;

response = userEntry.nextLine();

number = Integer.parseInt(response);

int counter = 0, attempt = 0;

//Paket yollamaya başlamadan önceki başlangıç zamanı

long startTime = System.nanoTime();

do {

message = "PCK";

this.sendMessage(message + counter);

attempt++;

String request = this.getRequest();

//Gelen mesajda "Port adı , mesaj " şeklinde geldiğinden port adı ile mesajı ayırmak için split kullanıyor

String[] split = request.split(",");

str2 = split[split.length-1].substring(0, 3);

//Paket yerine ulaşmamış ise ekrana yazı bastırılıyor tekrar kontrol ediliyor

while (!str2.equals("ACK")) {

System.out.println(message + counter + " Resending...");

output.println(message + counter);

attempt++;

split = this.getRequest().split(",");

if(split != null){

str2 = split[split.length-1].substring(0, 3);

System.out.println(str2);

}

}

System.out.println(request + " received from receiver successfully");

counter++;

} while (counter < number);

long endTime = System.nanoTime();

System.out.println("Total number of try: " + attempt);

System.out.println("System afficiency: " + (double) number/attempt);

System.out.println("Time taken to send all packets: " +(endTime - startTime) + " nano seconds.");

}

public void sendMessage(String message){

output.println(message);

}

public String getRequest(){

return input.hasNext() ? input.nextLine() : "";

}

public void serviceStart(int PORT) {

try {

host = InetAddress.getLocalHost();

link = new Socket(host, PORT);

input = new Scanner(link.getInputStream());

output = new PrintWriter(link.getOutputStream(), true);

} catch (Exception uhEx) {

System.exit(1);

}

}

}

//SENDER(GÖNDERİCİ)

Public Sender (int port ){

this.PORT=port #burada dinleyeceği portun numarasını veriyor

host = InetAddress.getLocalHost();

link = new Socket(host, PORT);

input = new Scanner(link.getInputStream())

output = new PrintWriter(link.getOutputStream(), true)

//local hostun bilgisini alıp soket oluşturuyor. PrintWriter yazmak için yazıcı oluşturuyor.

Sonra tekrardan run fonksiyonun içine giriyor ve server erişim sağlıyor. Kaç tane paket yollamak istiyorsun diye soruyor Scanner ile beraber kullanıcıdan bilgi alıyor.

response = userEntry.nextLine()

number = Integer.parseInt(response);

int counter = 0, attempt = 0

//burada ise kullanıcıdan aldığı bilgiyi nextline ile alıyor ve response içine atıyor.

İnteger ifadeyi numbere dönüştürüyor.

Attempt # bunu kayıp olarak yapıyor.

//Paket yollamaya başlamadan önceki başlangıç zamanı

long startTime = System.nanoTime()

this.sendMessage(message + counter)

// dinlediği porta mesajı ve sayacı yolluyor

attempt++

//bir arttıyor

ROOT

package Root;

//java.io ile veri akımını sağlayan ilk paket

import java.io.IOException;

//Routerden bilgi yazabilmek için PrintWriter

import java.io.PrintWriter;

import java.net.InetAddress;

import java.net.ServerSocket;

import java.net.Socket;

import java.util.Scanner;

import java.util.logging.Level;

import java.util.logging.Logger;

//extends ile Threadden miras kalan özellikleri yazıyoruz.

public class Root extends Thread{

protected ServerSocket serverSocket;

//LISTENPORT ile port numarasını alıyoruz.

protected int LISTENPORT;

protected InetAddress host;

//synchronized metodu ile de Threadler de sıra ile girmelerini ve biri bitmeden diğerini girdirmiyor.

protected synchronized void openListenPort(int PORT){

this.LISTENPORT = PORT;

System.out.println("Opening port");

try {

//Local ip bilgileri alınıyor

//InetAddress ile nesnenin ham IP adresini döndürüyor.

// getLocalHost ile de yerel bilgisayarın adını döndürüyor.

host = InetAddress.getLocalHost();

//Dinlenilecek port serverSockete set ediliyor

//Soketi açıyor Router buraya erişiyor.

serverSocket = new ServerSocket(LISTENPORT); //Step 1.

// host = InetAddress.getLocalHost(); ile serverSocket = new ServerSocket(LISTENPORT); ile dinlenilcek portu burada açıyor.

} catch (IOException ioEx) {

System.out.println(

"Unable to attach to port for receiver!");

System.exit(1);

}

}

// Gönderim için Rsender sınıfını kullanıyor.

//Gönderebilmek için soket oluşturuyoruz.

public class Rsender{

private Scanner sendInput;

private PrintWriter sendOutput;

private Socket socket;

public Rsender(Socket socket){

this.socket = socket;

try {

//getınputStream ile istek kabul edildikten sonra soket sınıfında yer alan metod ile istemciden gelen veriler okunmuştur.

sendInput = new Scanner(this.socket.getInputStream());

//getOutStream ile istemciye bir veri göndermek istendiğinde kullanıyoruz.

sendOutput = new PrintWriter(this.socket.getOutputStream(), true);

} catch (IOException ioEx) {

System.err.println("Root.Router.startSenderService()");

System.exit(1);

}

}

//mesaj+sayaç gönderme de sendMessage kullanıyoruz.

public void sendMessage(String message){

sendOutput.println(message);

}

public String getRequest(){

if (sendInput.hasNext()) {

return sendInput.nextLine();

}

return null;

}

public boolean hasMessage(){

return sendInput.hasNext();

}

public void closeConn(){

// try-catch metodlarını da hataları yakalamak için kullanıyoruz.

try {

socket.close();

} catch (IOException ex) {

//bağlantıyı kesmek çin Receiverden komutu alıyor.

Logger.getLogger(Root.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

}