

SUNUCU YAZILIM TEKNOLOJİLERİ

PROJE 1 RAPORU

Konu: Asenkron Mesajlaşma için Soketler Kullanılarak Bir
İletişim Altyapısı Gerçekleştirimi

Hazırlayan:
05110000044 – Onurhan ÇELİK

Sunulan:
Doç. Dr. Rıza Cenk ERDUR

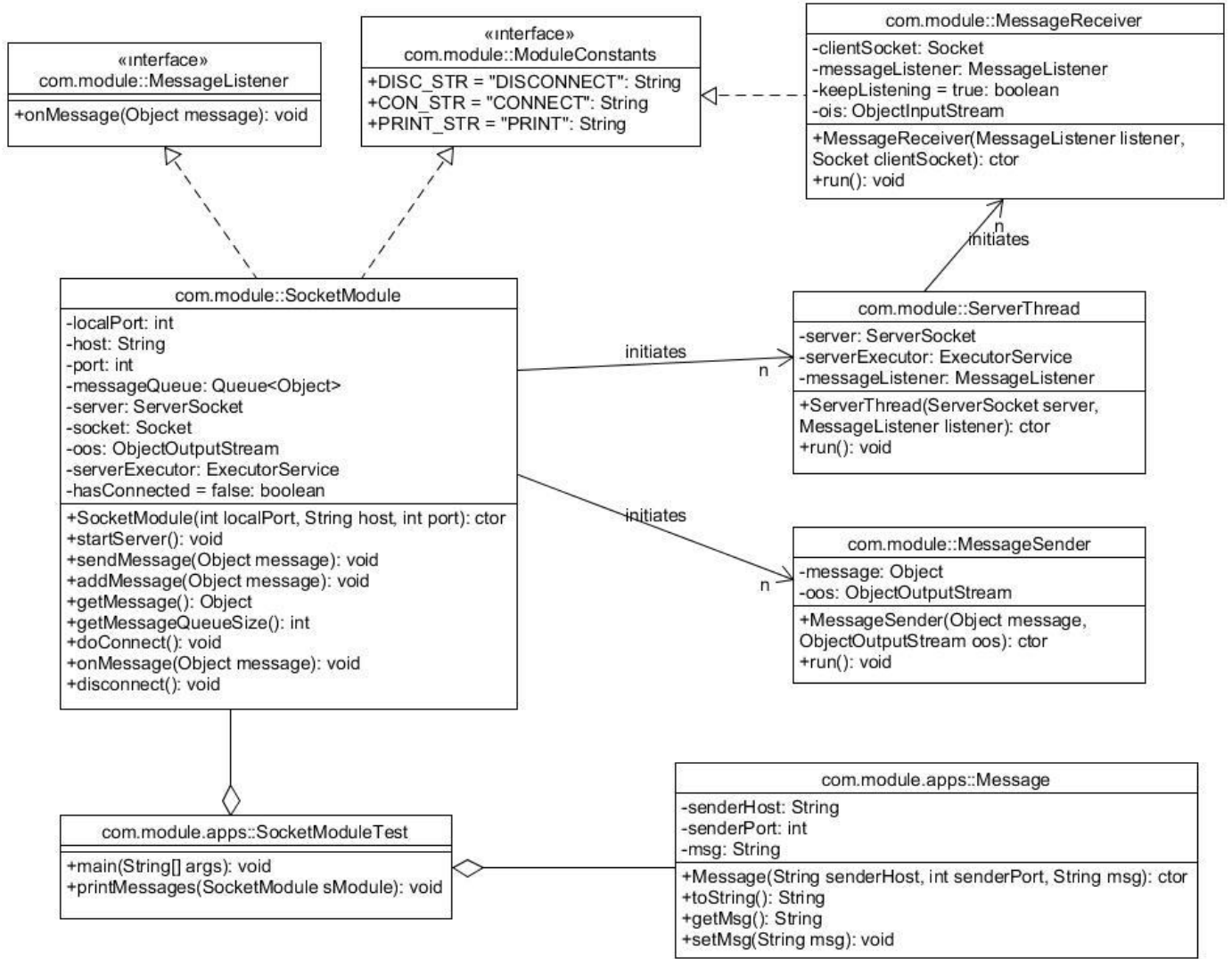
14 Nisan 2016
Bornova, İZMİR

İÇİNDEKİLER

1. TASARIM.....	2
1.1. “com.module” Paketindeki Sınıflar.....	3
1.2. “com.module.apps” Paketindeki Sınıflar	8
2. KULLANICI KILAVUZU	11

1. TASARIM

Şekil 1’de projede var olan sınıfların UML gösterimi mevcuttur. SocketModule sınıfı iki farklı arayüz sınıfını gerçekleştirmektedir. Ayrıca ServerThread ve MessageSender thread’lerini başlatmaktadırlar. ServerThread sınıfı ise MessageReceiver thread’ini başlatmaktadır. SocketModuleTest sınıfı ise main() metodunu barındıran test uygulaması sınıfıdır ve SocketModule ve Message sınıflarından nesne yaratıp kullanmaktadır.



Şekil 1. UML Sınıf Diyagramı

- **Projedeki paketler**

Projenin temelinde “**com.module**” paketi vardır ve bu paket altında da “**com.module.apps**” paketi yer almaktadır. “module” paketindeki sınıflar iletişim altyapısını oluşturan sınıflardır. “module.apps” paketinde ise bu iletişim altyapısını test etmek için yapılan uygulama ve kullanılan sınıflar yer almaktadır.

1.1. “com.module” Paketindeki Sınıflar

- **SocketModule Sınıfı**

Bir uygulamanın yapılan iletişim altyapısını kullanması için bu sınıftan bir nesne üretmesi gerekir. Nesne oluştururken sınıfın yapıcı metodu üç tane parametre alır. Bu parametreler; uygulamanın server olarak dinleyeceği port numarası(*int localPort*), uygulamanın mesaj gönderirken kullanacağı ip adresi(*String host*) ve port numarasıdır(*int port*).

- **public void startServer():** Bu metot ServerSocket nesnesini yaratır ve server olarak belirlenen portun dinlemesi için ServerThread iş parçacığını çalıştırır. Bu metot SocketModule nesnesi yaratıldıktan sonra bir kere çağrılır uygulama içinde.

```
public void startServer() {
    try {
        server = new ServerSocket(localPort);
        System.out.println("Server is now listening "+localPort+"
port.");
        serverExecutor.execute(new ServerThread(server, this)); //runs
ServerThread thread to listen connections and handle events
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

- **public void sendMessage(Object message):** Client olarak mesaj göndermek için bu metot uygulamalar tarafından kullanılır. Mesajın tipi, esnek bir yapı olması amacıyla Object sınıfıdır. Bu metot kendi içinde mesajın yollanması için MessageSender iş parçacığını çalıştırır. Mesaj göndermek için client'ın server'a bağlı olması gerekir. Sınıfta var olan boolean tipindeki hasConnected değişkeni bu bağlantının kurulu olup olmadığını belirtir. Mesaj göndermeden önce hasConnected değeri kontrol edilir. Eğer bağlantı kurulu değilse doConnect() metodu çağrılır, daha sonra mesaj göndermek için MessageSender iş parçacığı çalıştırılır.

```
public void sendMessage(Object message) {
    if(!hasConnected) //we need to be connected before send
        doConnect();
    if(hasConnected)
        serverExecutor.execute(new MessageSender(message, oos)); }
```

- **public void addMessage(Object message):** SocketModule sınıfında yer alan Oueue<Object> tipindeki messageQueue adlı attribute gelen mesajları liste halinde tutmaktadır. Bu metot yeni gelen mesajın listeye eklenmesini sağlar.

```
//Add new message to end of the message queue
public void addMessage(Object message) {
    messageQueue.add(message);
}
```

- **public Object getMessage():** Bu metot mesaj listesindeki (messageQueue) son elemanı döndürür ve listeden siler. Liste boşsa null döner.

```
//Removes the first item in the queue and returns it
public Object getMessage() {
    if(messageQueue.peek() != null)
        return messageQueue.remove();
    else return null;
}
```

- **public int getMessageQueueSize():** Mesaj listesindeki mesaj sayısını döndürür.

```
//Returns the number of messages in the queue
public int getMessageQueueSize() {
    return messageQueue.size();
}
```

- **public void doConnect():** Client'ın mesaj göndermesi için server'a bağlı olması gerekir. Bu bağlantının kurulması bu metot tarafından yapılır. Socket nesnesi yaratılır ilgili port numarası ve host adresleriyle. Bu açılan socket'in output stream'i sınıfın bir değişkeni olan ObjectOutputStream(oos) nesnesine verilir. Bağlantının gerçekleştiğini belirtmek için hasConnected değişkeni true olarak ayarlanır.

```
//Before send the message, we need to connect other client
public void doConnect()
{
    try {
        socket = new Socket(InetAddress.getByName(host),port);
        oos = new ObjectOutputStream(socket.getOutputStream());
        hasConnected = true; //status flag is updated
    } catch (ConnectException e) {
        System.out.println("ServerSocket is not open.");
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

- **public void onMessage(Object message):** MessageListener arayüz sınıfının metodu olan bu metod MessageReceiver iş parçacığı tarafından çağrılır. Yeni bir mesaj geldiğinde, mesajın işlenmesi için bu metod çağrılır. Bu metod yeni gelen mesajı listeye ekler. Listeye ekleme şeklinde yapılan mesajı işleme basit tutulmuştur. İstenilen gereksinime göre bu metod geliştirilebilir.

```
public void onMessage(Object message) {
    messageQueue.add(message); //adds incoming message to message
    queue.
}
```

- **public void disconnect():** Eğer uygulama mesaj göndermek için açtığı client socket bağlantısını kesmek isterse bu metod çağrılır. MessageSender iş parçacığı çağrılır bağlantının kesildiğini server'a iletmek için, ancak bu sefer mesaj olarak “DISCONNECT” string’i yollanır.

```
/*
 * When application sends the disconnect message, connection is
 * terminated with server.*/
public void disconnect()
{
    if(hasConnected)
    {
        serverExecutor.execute(new MessageSender(DISC_STR, oos));
        hasConnected=false; //connection is closed with server
    }
}
```

• MessageListener Arayüz Sınıfı

Bu arayüz sınıfı yeni gelen mesajları almak için kullanılır. Arayüz sınıfını gerçekleştirecek sınıflar “*public void onMessage(Object message)*” metodunu içermelidirler.

```
public interface MessageListener {
    //receive new message
    public void onMessage(Object message);
}
```

• ServerThread Sınıfı

Bu sınıf Runnable arayüz sınıfını gerçekleştirir ve iş parçacığı olarak görev alır. Bu iş parçacığı sürekli çalışır. Server’a gelen bağlantıları dinler ve bağlantı açıldığında geldiğinde MessageReceiver iş parçacığını çalıştırır o bağlantıda gelecek olan mesajların işlenmesi için. Server’ın gelen bağlantıları kabul etmesi ayrı bir iş parçacığında yapıldı. Çünkü sürekli çalışması gerekiyor ve ana sınıfta bu işlem yapılsaydı program donardı.

Sınıfın yapıcı metodu ServerSocket nesnesini alır bu nesne üzerinden bağlantı kabul edilir. Ayrıca MessageListener nesnesi alır, bağlantı geldiğinde mesajların dinlenmesinde kullanılmak üzere.

İş parçacığının run() metodu aşağıdaki tabloda görülebilir.

```
@Override
public void run() {
    try {
        while (true) {
            Socket clientSocket = server.accept(); //accepts incoming
            connections and creates client socket
            System.out.println("Connection was established with client.");

            serverExecutor.execute(new MessageReceiver(messageListener,
            clientSocket)); //runs the MessageReceiver thread to handle incoming
            message
        }
    } catch (SocketException e) {
        System.out.println(e.getMessage());
    } catch (IOException e) {
        System.out.println(e.getMessage());
        e.printStackTrace();
    }
}
```

- **MessageSender Sınıfı**

Bu sınıf Runnable arayüz sınıfını gerçekleştirir ve iş parçacığı olarak görev alır. Client'ın gönderdiği mesajları server'a iletir ayrı bir thread mekanizması içinde. Yapıcı metodu Object tipindeki gönderilecek mesajı ve mesajın gönderileceği ObjectOutputStream nesnesini alır.

İş parçacığının run() metodu aşağıdaki tabloda görülmektedir.

```
@Override
public void run() {
    try {
        oos.writeObject(message); //writes message to stream
        oos.flush();
    } catch (IOException e) {
        //e.printStackTrace();
        System.out.println("Server is disconnected.");
    }
}
```

- **MessageReceiver Sınıfı**

Bu sınıf Runnable arayüz sınıfını gerçekleştirir ve iş parçacığı olarak görev alır. Bu iş parçacığı Server tarafından her client bağlantı için ayrı bir thread olarak çalıştırılır, böylece eş zamanlı olarak birden fazla client'tan gelen mesajları işleyebilir. Server'a her yeni bir bağlantı geldiğinde bu thread çalıştırılır. Bu thread'in çalışması client ile olan bağlantı sonlanana kadar devam eder. MessageReceiver tek bir client'tan gelen mesajları dinler ve bu mesajları Server'a geçirir onMessage metodu ile. Her gelen mesaj bu metot tarafından ayrı zamanlarda işlenir. Bu yapı asenkron mesajlaşmayı sağlar.

Bağlantı boyunca input stream sürekli dinlenir. Eğer stream'den alınan mesaj null değilse mesaj incelenir. Mesajlar Object tipinde gelmektedir. Bu mesajların sınıfının toString() metodu kontrol edilir. Eğer bu metot "DISCONNECT" string'i döndürdüyse, client bağlantının sonlandığını belirtmiştir. Artık server'ın bu client ile olan bağlantıyı dinlemesine gerek yoktur. Input stream ve socket kapatılır.

MessageReceiver sınıfının yapıcı metodu MessageListener tipinde değişken almaktadır. Yeni mesajları bu listener'ın onMessage metoduna geçirecektir. Aldığı diğer değişken ise Socket nesnesidir. Bağlı olan client'ın bağlantı bilgisini almak için kullanılır.

MessageReceiver iş parçacığı gelen mesajları mesaj listesine atılması işlemi dışında ayrıca her mesajı, kullanıcı ekranına yazdırır. Bu yazdırmadaki string değeri Object tipindeki mesajın toString() metodundan alınır.

MessageReceiver sınıfının override ettiği run() metodunun kodu aşağıdaki tabloda görülebilir.

```
@Override
public void run() {
    while (keepListening) {
        Object message = null; //creates incoming message object
        try {
            message = ois.readObject(); //reads the input stream
        } catch (IOException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }

        if (message != null) {
            if((message.toString()).equalsIgnoreCase(DISC_STR))
//if message is disconnect message
            {
                keepListening=false; //stop listening
because client has been disconnected
            }
            else{
                //send message to message listener
                messageListener.onMessage(message);
            }
        }
    }
}
```



```

        System.out.println();
        System.out.println("Incoming message: " +
message.toString()); //writes incoming message to console instantly
    }
}

try {
    ois.close(); //closes the input stream - stop
listening this socket
} // end try
catch (IOException ioException) {
    ioException.printStackTrace();
} // end catch
}
}

```

• ModuleConstants Arayüz Sınıfı

Bu arayüz sınıfı hem iletişim modülündeki hem de uygulamalardaki farklı sınıflar tarafından kullanılan sabitleri içermektedir. Sınıflar bu static değişkenlere *static import* kullanarak erişebilirler.

- *public static final String DISC_STR = "DISCONNECT"* : Bağlantının sonlandırılacağını bildiren sabit değişken.
- *public static final String CON_STR = "CONNECT"* : Bağlantı kurulacağını bildiren sabit değişken.
- *public static final String PRINT_STR = "PRINT"* : Uygulamalar tarafından kullanılan ve mesajların ekrana yazdırılacağını bildiren sabit değişken.

1.2. “com.module.apps” Paketindeki Sınıflar

• Message Sınıfı

Bu sınıf iletişim altyapısını kullanacak olan uygulamaya özgü test amaçlı oluşturulan bir sınıftır. İletişimde gönderilecek olan mesajların yapısal olması amacıyla kullanılır. Sınıf değişkenleri olarak mesajın göndericisinin bilgisi amacıyla host adresi ve port numarası vardır. Ayrıca mesaj olarak temelde bu sınıf String tipinde bir değişken içermektedir. Başka uygulamalarda örneğin dosya paylaşımı gibi gönderilecek mesajın yapısına göre bu mesaj içerik tipi değiştirilebilir. Message sınıfından nesne oluşturmak için yapıcı metot bu üç değişkeni içermektedir.

Ayrıca Message sınıfı toString() metodunu override etmektedir. Yapılan test uygulaması gereği mesajın konsola yazdırılması amacıyla bu metod override edilmiştir. String tipinde gönderici bilgisi ve mesajı döndürür. Bu bilgiler arasına ">>>" işaretini ekler. Bu metodun kodu aşağıdaki tabloda görülebilir.

```
@Override
public String toString() {
    return new String(senderHost+":"+senderPort+">>>"+msg);
}
```

• SocketModuleTest Sınıfı

Bu sınıf, test uygulamasının main metodunun da içinde yer aldığı sınıftır. Test uygulaması konsol ekranında çalışmaktadır. Çalıştırılması için Main metoduna değişken parametreleri verilmesi gerekmektedir(*String[] args*). Bu parametre değişkenleri sırasıyla server olarak dinlenecek port numarası, client olarak yaratılacak socket'in host adresi ve port numarasıdır. Bu değişkenlerin atanmasının kod parçası aşağıdaki tabloda görülebilir.

```
// localPort=args[0] , host=args[1] , port=args[2]
int localPort=Integer.parseInt(args[0]); //listens this port number as
server
String host = args[1]; //host-server address
int port = Integer.parseInt(args[2]); //sends message from this port as
client
```

Alınan bu değişkenler kullanılarak SocketModule sınıfından bir nesne üretilir. Daha sonra SocketModule sınıfının startServer() metodu çağrılır. Böylece uygulama server olarak dinleme konumuna geçmiş olur. Bu kısmın kod parçası aşağıdaki tabloda görülebilir.

```
SocketModule sModule = new SocketModule(localPort,host,port);
sModule.startServer(); //server started
```

Daha sonra while(true) döngü yapısı mevcuttur uygulamada. Sürekli olarak gönderilecek olan mesajın string girdisi kullanıcıdan alınır. Ancak kullanıcı çeşitli anahtar kelimeler girerse uygulama mesaj göndermenin dışında farklı işlemler de gerçekleştirebilir. Bu yapı if-else bloğu ile oluşturulmuştur.

Eğer kullanıcı "DISCONNECT" string değerini konsolda girdiyse SocketModule sınıfının disconnect() metodu çağrılır. Böylece uygulama daha önce eğer bağlandıysa server'a, bağlantısı kesilir ve mesaj gönderemez duruma gelir. Ancak bu "disconnect" durumunda uygulamanın server özelliği hala çalışır durumdadır ve ilgili port'u bağlantı veya mesaj gelmesine karşı dinlemektedir. Uygulama sonlandırılana kadar(System.exit) bu devam eder. Eğer kullanıcı "disconnect" durumundaysa tekrardan server'a bağlanmak için "CONNECT" string değerini girdi olarak ekrana girmelidir.

Kullanıcıdan sürekli string girdisi alıp yapılan mesaj gönderme veya diğer işlemlerin olduğu while döngüsünün kod parçası aşağıdaki tablodadır.

```
Scanner scanner = new Scanner(System.in);
while (true) { //loop continues until application is closed
    while (true) { //loop continues until taking disconnect message
        from user
        System.out.print("Message: ");

        String inputString = scanner.nextLine(); //message string
        value is taken from user

        if (inputString.equalsIgnoreCase(DISC_STR)) { //if string is
            disconnect message,
            sModule.disconnect(); //disconnect method will be call
            break; //break from inner while
        } else if (inputString.equalsIgnoreCase(PRINT_STR)) //if
            string is print message,
            printMessages(sModule); //print method will be call
        else {
            Message msg = new Message(host,localPort,inputString);
            //message object that will be send is created with sender info
            sModule.sendMessage(msg); //sends the message
            System.out.println();
        }
    }

    String msg2;
    do {
        System.out.println("Type \"CONNECT\" command to continue or
        exit program... ");
        System.out.print(">");
        msg2 = scanner.nextLine();
        if (msg2.equalsIgnoreCase(PRINT_STR))
            printMessages(sModule);
    } while (!msg2.equalsIgnoreCase(CON_STR)); //until taking connect
    string, input string will be taken from the user

    sModule.doConnect(); //opens the connection with server to send
    message
}
```

Ayrıca kullanıcı bağlı olma veya bağlı olmama durumuna bakmaksızın “PRINT” string değerini ekrana girerse şimdiye kadar aldığı mesajların listesinin dökümünü kendi uygulama ekranında görebilir. Bu işlem sonrası kendi mesaj listesi temizlenmiş olur. Bu yazdırma işlemini yapan metot aşağıdaki tabloda görülebilir.

```
public static void printMessages(SocketModule sModule)
{
    Object msg;
    System.out.println("List of all received messages:");

    if(sModule.getMessageQueueSize()==0) //if there is no message in
the message queue
        System.out.println("There are no messages to be
displayed.");
    else{
        while( (msg=sModule.getMessage())!=null) //gets message from
the queue until it is empty
        {
            System.out.println(msg.toString()); //prints the
message to the console
        }
    }
}
```

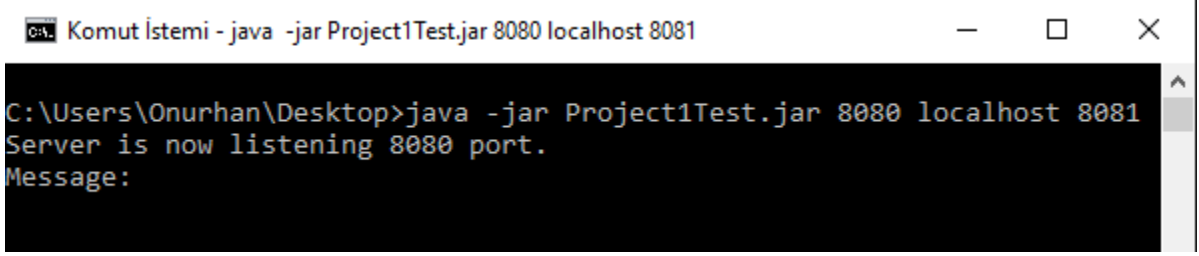
Uygulama ilk başladığında client olarak bağlı durumda değildir ve ilk aşamada eğer özel anahtar girdileri dışında bir mesaj girerse otomatikman bağlı konumuna gelir.

Kullanıcıdan sürekli alınabilen mesaj girdileri ile mesajlar eğer bağlı server varsa server’a iletilir. Mesajı alan uygulama ekranında ayrıca mesaj otomatikman ekrana yazdırılır.

2. KULLANICI KILAVUZU

Uygulamayı çalıştırmak için proje dosyalarından Export edilen Project1Test.jar java dosyasının komut penceresinden(cmd.exe) parametre değerleri(server port, client host, client port) verilerek çalıştırılması gerekir. Örnek girdi olarak komut penceresinde Project1Test.jar dosyasının konumuna gelindikten sonra “**java -jar Project1Test.jar 8080 localhost 8081**” bu komut girilirse, uygulamayı çalıştırılan cihaz üstünden bir server açılmış olur. Bu server 8080 portunu dinler. Ayrıca uygulamanın mesaj göndermesi “localhost” adresi üzerinden 8081 portu ile gerçekleşir. Kullanıcı kılavuzu hazırlarken test için iki farklı uygulama için Test uygulamasını iki kez farklı parametrelerle başlattım. Bu iki uygulama arasında iletişim oluşturulmuştur.

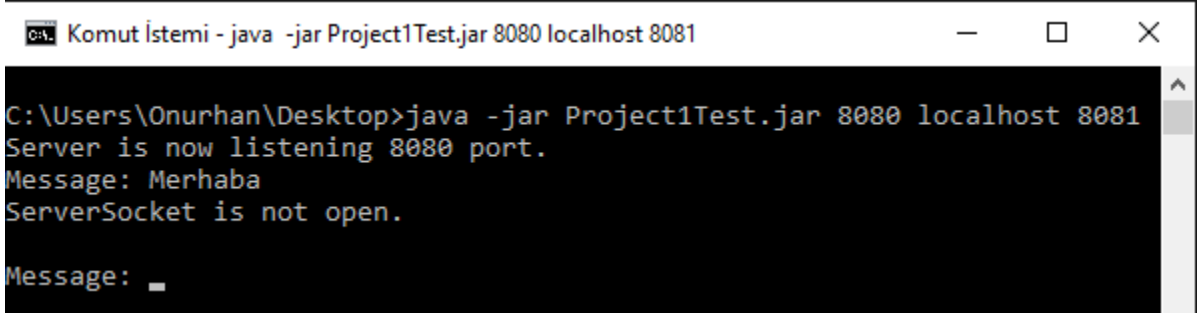
Şekil 2’de komut penceresinde uygulamayı örnek verilerle başlatmayı ve daha sonra ilk gelen ekran görülebilir. Uygulama başlatılınca server olarak dinlemeye başlar. Ayrıca kullanıcıdan mesaj girdisi ister.



```
Komut İstemi - java -jar Project1Test.jar 8080 localhost 8081
C:\Users\Onurhan\Desktop>java -jar Project1Test.jar 8080 localhost 8081
Server is now listening 8080 port.
Message:
```

Şekil 2. Komut penceresinden uygulamanın başlatılması

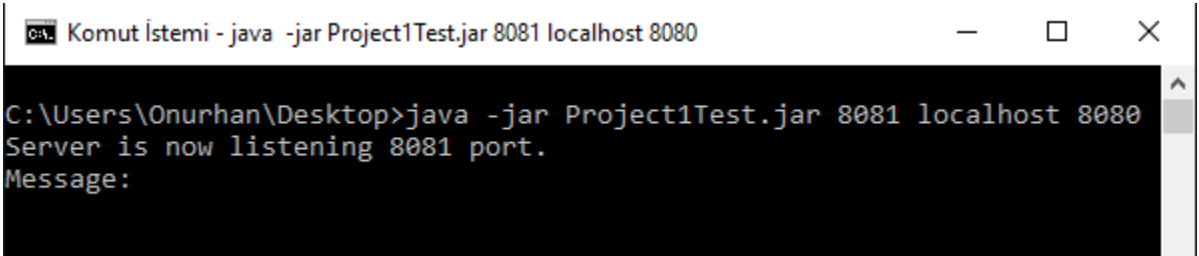
Kullanıcı mesaj girdiğinde eğer uygulamanın mesaj gönderdiği server kapalıysa Şekil 3’teki gibi bir uyarı mesajı gelir. Server’ın açık olmadığı geri dönüşünü alır.



```
Komut İstemi - java -jar Project1Test.jar 8080 localhost 8081
C:\Users\Onurhan\Desktop>java -jar Project1Test.jar 8080 localhost 8081
Server is now listening 8080 port.
Message: Merhaba
ServerSocket is not open.
Message: _
```

Şekil 3. Mesaj gönderilen server’ın kapalı olması durumu

Yukarıda başlatılan uygulamayla iletişimi sağlamak için farklı bir komut penceresi daha açılarak, farklı parametre verileriyle yeni bir uygulama başlatılmasını Şekil 4’te görebilirsiniz. Burada uygulama server olarak 8081 portunu dinlerken, mesaj göndermek için localhost adresini ve 8080 port numarasını kullanmaktadır.



```
Komut İstemi - java -jar Project1Test.jar 8081 localhost 8080
C:\Users\Onurhan\Desktop>java -jar Project1Test.jar 8081 localhost 8080
Server is now listening 8081 port.
Message:
```

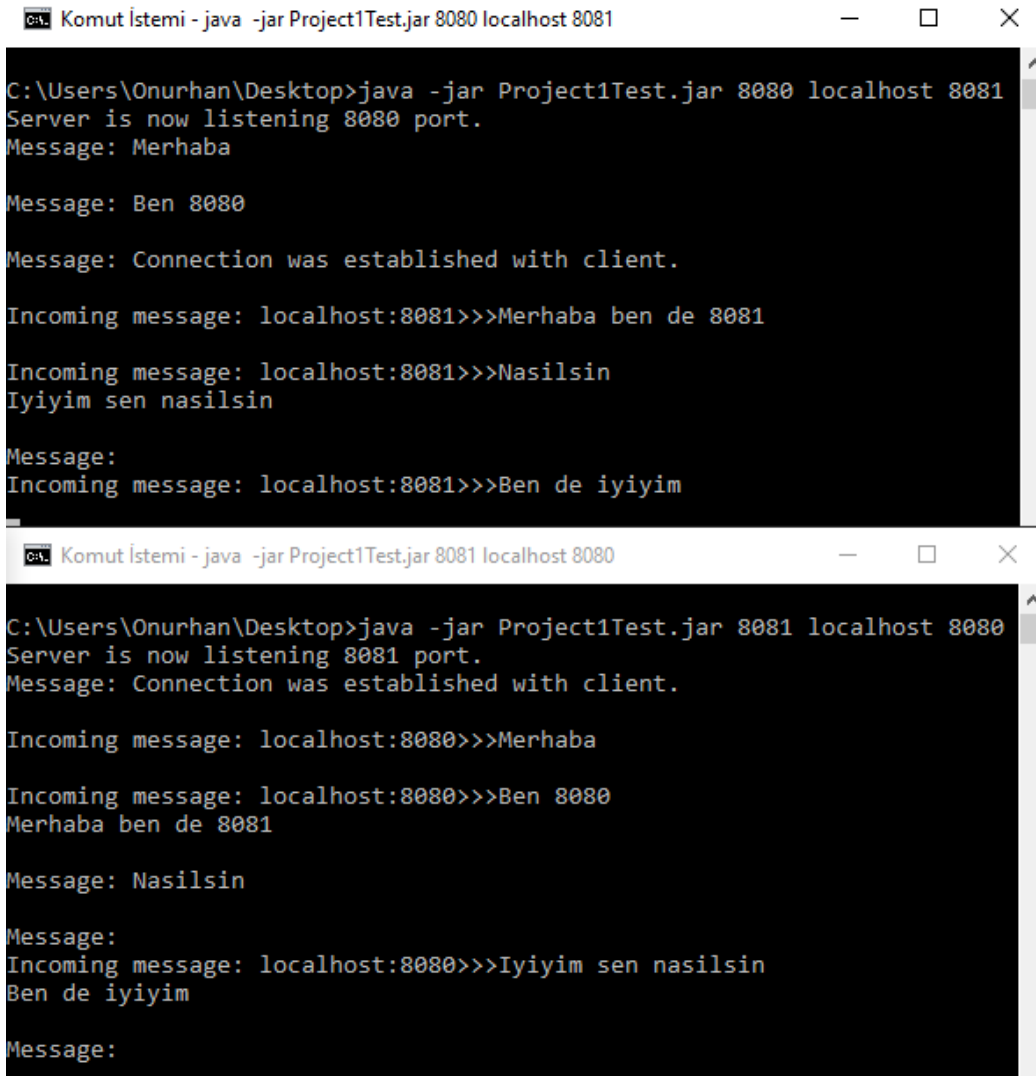
Şekil 4. İkinci uygulamanın yeni bir komut penceresinde farklı değerlerle başlatılması

Artık birbiriyle mesaj alışverişinde bulunacak iki uygulama da başlatılmış oldu. Her iki uygulama da server olarak farklı portları dinlemektedir. Eğer farklı makinalarda/ağlarda bu uygulamalar başlatılsaydı dinledikleri port numaraları aynı olabilirdi. Ancak aynı yerel makinada

bu test işlemi yapıldığı için dinledikleri port numaraları farklı olmak zorundadır. İlk başlatılan uygulama 8080 portunu dinleyip , localhost:8081'e mesaj gönderecektir. İkinci başlatılan uygulama ise server olarak 8081 portunu dinleyip, mesajlarını localhost:8080'e gönderecektir.

Uygulamalar birbirlerine cevap beklemeden sürekli ve istedikleri zaman mesaj gönderebilirler. Yani asenkron mesajlaşmayı uygulamaların kullandığı iletişim altyapısı desteklemektedir.

Uygulamaların birbirlerine üst üste ve sırayla attıkları mesajları gösteren durumlar Şekil 5'te görülebilir. Üstteki pencere ilk başlatılan uygulama(8080 server'ı), alttaki uygulama ise ikinci başlatılan uygulamadır(8081 server'ı). Kullanıcılardan uygulama penceresine yazdıkları girdi sonrası tekrardan yeni mesaj girdisi istenir. Ayrıca alınan mesajlar anlık olarak uygulama penceresinde yazdırılır. Mesaja ek olarak gönderen uygulamanın adres bilgisini de içerir yazdırma işlemi.

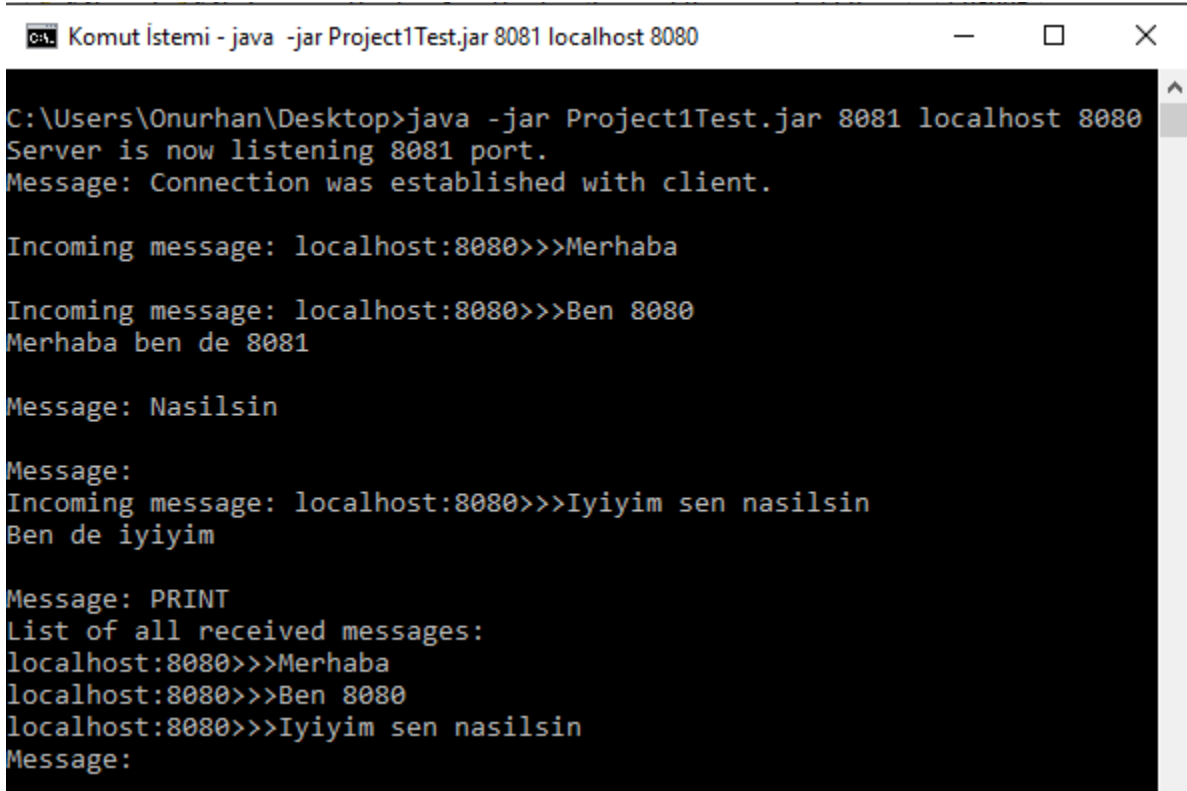


```
Komut İstemi - java -jar Project1Test.jar 8080 localhost 8081
C:\Users\Onurhan\Desktop>java -jar Project1Test.jar 8080 localhost 8081
Server is now listening 8080 port.
Message: Merhaba
Message: Ben 8080
Message: Connection was established with client.
Incoming message: localhost:8081>>>Merhaba ben de 8081
Incoming message: localhost:8081>>>Nasilsin
Iyyim sen nasilsin
Message:
Incoming message: localhost:8081>>>Ben de iyyim

Komut İstemi - java -jar Project1Test.jar 8081 localhost 8080
C:\Users\Onurhan\Desktop>java -jar Project1Test.jar 8081 localhost 8080
Server is now listening 8081 port.
Message: Connection was established with client.
Incoming message: localhost:8080>>>Merhaba
Incoming message: localhost:8080>>>Ben 8080
Merhaba ben de 8081
Message: Nasilsin
Message:
Incoming message: localhost:8080>>>Iyyim sen nasilsin
Ben de iyyim
Message:
```

Şekil 5. Birbiriyle iletişimde olan iki farklı uygulamanın mesaj alışverişi sonrası ekran durumları

Daha sonra kullanıcı girdisi olarak anahtar girdi değerlerinden olan “**PRINT**” girdisini ikinci programda girilmesi sonucu gelen durumun ekran görüntüsü Şekil 6’da görülmektedir. Daha önce aldığı ve hala mesaj listesinde bulunan mesajlar toplu ve sıralı şekilde ekranda listelenmiştir. Bu anahtar değeri girildiğinde test uygulaması mantığında herhangi bir mesaj iletişim kanalına alınmıyor, sadece tek taraflı bir işlem söz konusudur. Bu şekilde iletişim bağlantısı içinde olan uygulama farklı bir iş yapmaktadır, bu farklı işi yapma sırasında hala server olarak mesaj ve bağlantılar için dinleme durumundadır.



```
Komut İstemi - java -jar Project1Test.jar 8081 localhost 8080

C:\Users\Onurhan\Desktop>java -jar Project1Test.jar 8081 localhost 8080
Server is now listening 8081 port.
Message: Connection was established with client.

Incoming message: localhost:8080>>>Merhaba

Incoming message: localhost:8080>>>Ben 8080
Merhaba ben de 8081

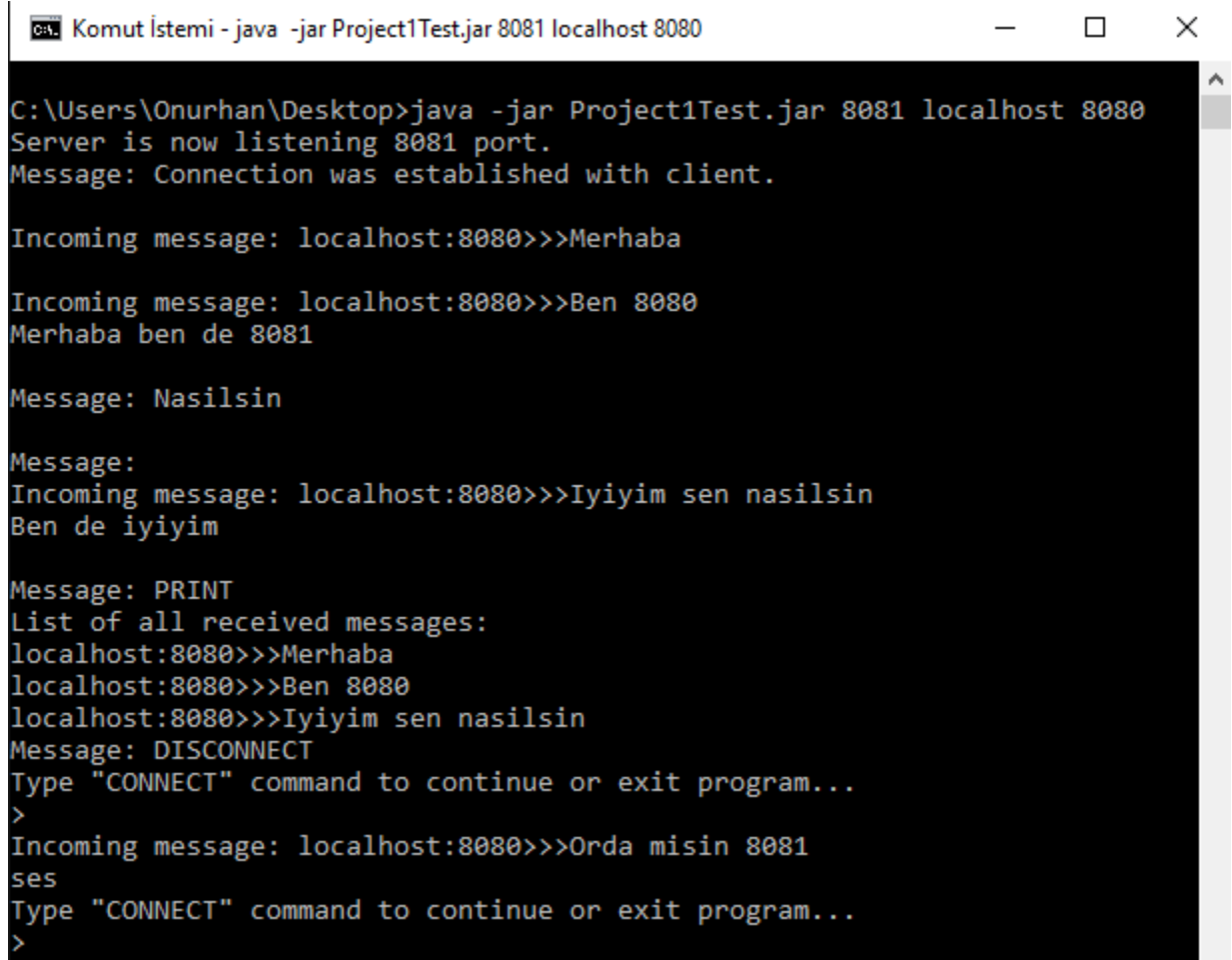
Message: Nasilsin

Message:
Incoming message: localhost:8080>>>Iyyim sen nasilsin
Ben de iyyim

Message: PRINT
List of all received messages:
localhost:8080>>>Merhaba
localhost:8080>>>Ben 8080
localhost:8080>>>Iyyim sen nasilsin
Message:
```

Şekil 6. Girdi olarak “PRINT” anahtar girdi değeri girilmesi sonucu oluşan ekran durumu

Bir diğer anahtar verisi olan **“DISCONNECT”** girdisi ikinci uygulama ekranında girilirse eğer bu uygulamanın client olarak mesaj göndermesi devre dışı kalır. Ancak server olarak hala dinlemededir. Bu durumun görüntüsü Şekil 7’dedir. Şekilde “disconnect” mesajı girilen uygulama hala diğer uygulamanın gönderdiği mesajları almaktadır. Ancak kendisi mesaj gönderememektedir.



```
Komut İstemi - java -jar Project1Test.jar 8081 localhost 8080

C:\Users\Onurhan\Desktop>java -jar Project1Test.jar 8081 localhost 8080
Server is now listening 8081 port.
Message: Connection was established with client.

Incoming message: localhost:8080>>>Merhaba

Incoming message: localhost:8080>>>Ben 8080
Merhaba ben de 8081

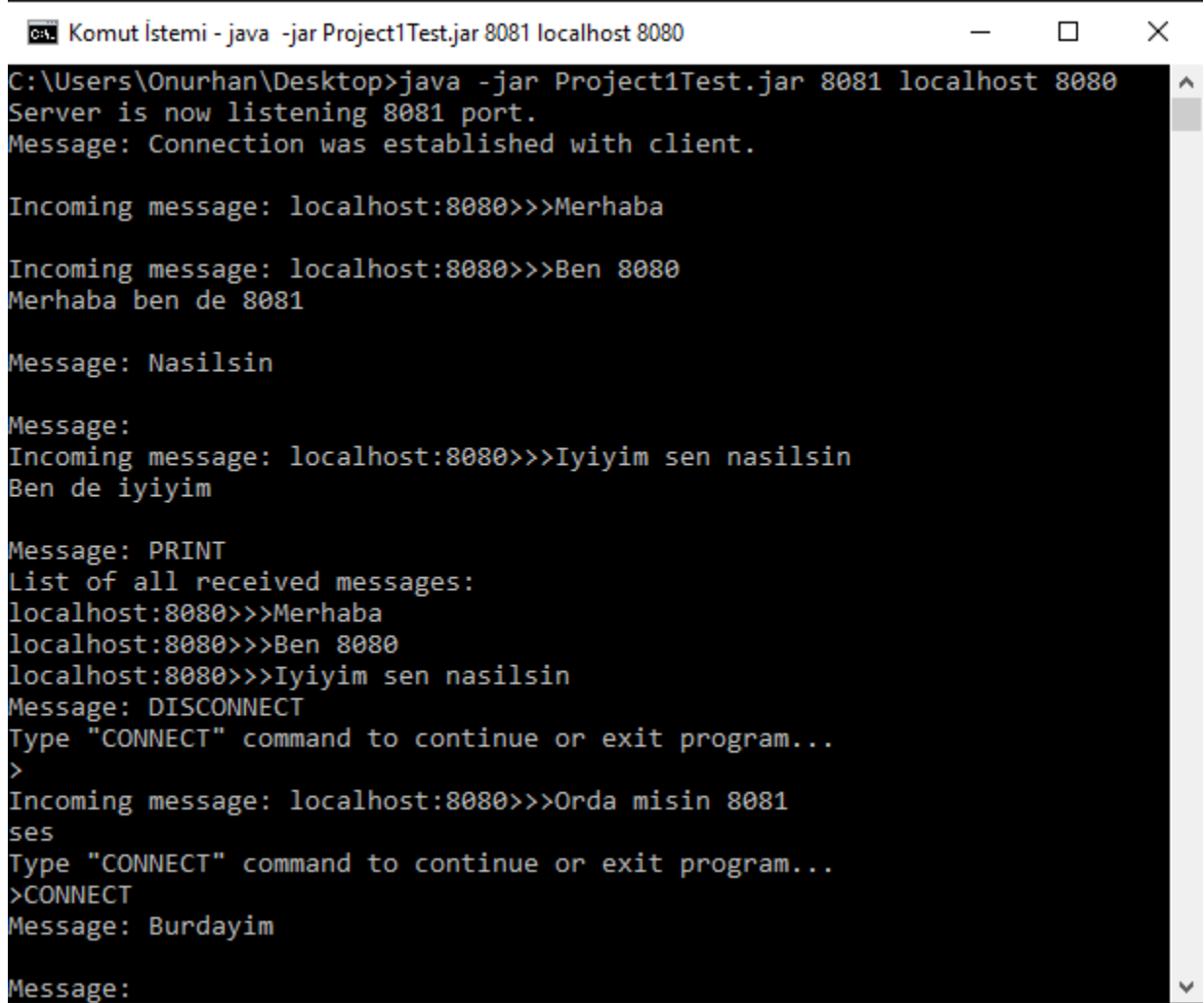
Message: Nasilsin

Message:
Incoming message: localhost:8080>>>Iyyim sen nasilsin
Ben de iyyim

Message: PRINT
List of all received messages:
localhost:8080>>>Merhaba
localhost:8080>>>Ben 8080
localhost:8080>>>Iyyim sen nasilsin
Message: DISCONNECT
Type "CONNECT" command to continue or exit program...
>
Incoming message: localhost:8080>>>Orda misin 8081
ses
Type "CONNECT" command to continue or exit program...
>
```

Şekil 7. “DISCONNECT” anahtar girdi değeri girilmesi sonrası oluşan durum

Bağlantı kopmuş şekildeyken kullanıcıdan alınan girdiler mesaj olarak gönderilemez. Tekrardan mesaj gönderme durumuna geçmesi için kullanıcının “**CONNECT**” anahtar girdi değerini girmesi gerekir. Böylelikle tekrardan client olarak mesaj gönderebilir. Bağlantının tekrar kurulması durumu Şekil 8’de görülebilir.



```
Komut İstemi - java -jar Project1Test.jar 8081 localhost 8080
C:\Users\Onurhan\Desktop>java -jar Project1Test.jar 8081 localhost 8080
Server is now listening 8081 port.
Message: Connection was established with client.

Incoming message: localhost:8080>>>Merhaba

Incoming message: localhost:8080>>>Ben 8080
Merhaba ben de 8081

Message: Nasilsin

Message:
Incoming message: localhost:8080>>>Iyyim sen nasilsin
Ben de iyyim

Message: PRINT
List of all received messages:
localhost:8080>>>Merhaba
localhost:8080>>>Ben 8080
localhost:8080>>>Iyyim sen nasilsin
Message: DISCONNECT
Type "CONNECT" command to continue or exit program...
>
Incoming message: localhost:8080>>>Orda misin 8081
ses
Type "CONNECT" command to continue or exit program...
>CONNECT
Message: Burdayim

Message:
```

Şekil 8. “CONNECT” anahtar girdi değeriyle mesaj göndermenin tekrar aktif olma durumu