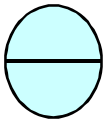
Soketler ve Java Soket Classları

- Soket, ağ üzerinde çalışan iki program arasındaki iki yönlü iletişim bağının ucudur.
- Soket, bir port numarasına bağlıdır. Bu, TCP katmanının, verinin gönderileceği uygulamayı tanıyabilmesi içindir.
- Her socket bir protokol ile eşleniktir (Ya UDP veya TCP)
- Java'nın .net paketi iki sınıf temin eder:
  - Socket –client tanımlamak için
  - ServerSocket –server tanımlamak için

Socket ve Port

- Client (Internet address=138.37.94.248)
  - Socket 
  - Any port(small int)
  - Other ports
- Server (Internet address=138.37.88.249)
  - Socket
  - Agreed port(small int)
  - Other ports
- To trasmit msg between a socket in one process (client) and a socket in another process(server)
- For a process to receive msg, its socket must be bound to a local port
- Any process can use multiple ports to receive msg, but no sharing of ports with other processes on same computer

UDP Programlama için Java API

- Java API, iki class aracılığıyla datagram iletişimi sağlar.
  - DatagramPacket: instance
    - One process ***send***s, other process ***receive***s
      - | Msg | length | Host | serverPort |
    - getData, getPort, getAddress to access
  - DatagramSocket
    - For UDP datagrams, constructor with arg port number: use particular port
    - No arg constructor: to choose a free local port
    - **send, receive methods** for transmitting datagrams
    - **setSoTimeout** to set a timeout
    - **connect:** connecting to a remote port and Inet address

Example: client and server

- Program for client
  - Creates a socket
  - Sends a msg to a server at port 6789
  - Waits to receive a reply
  - Args of main: msg and hostname of server
  - Msg is converted to array of bytes, hostname to Inet address

UDP Client: Bir mesaj gönderip cevap alır

```
import java.net.*;
import java.io.*;
public class UDPClient
{
    public static void main(String args[]){
        // args give message contents and server hostname
        DatagramSocket aSocket = null;
          try {
                aSocket = new DatagramSocket();
                byte [] m = args[0].getBytes();
                InetAddress aHost = InetAddress.getByName(args[1]);
                int serverPort = 6789;
                DatagramPacket request = new DatagramPacket(m,  args[0].length(), aHost, serverPort);
                aSocket.send(request);
                byte[] buffer = new byte[1000];
                DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
                aSocket.receive(reply);
                System.out.println("Reply: " + new String(reply.getData()));
          }
          catch (SocketException e){System.out.println("Socket: " + e.getMessage());}
          catch (IOException e){System.out.println("IO: " + e.getMessage());}
          finally
        {
            if(aSocket != null) aSocket.close();
        }
    }
}
```

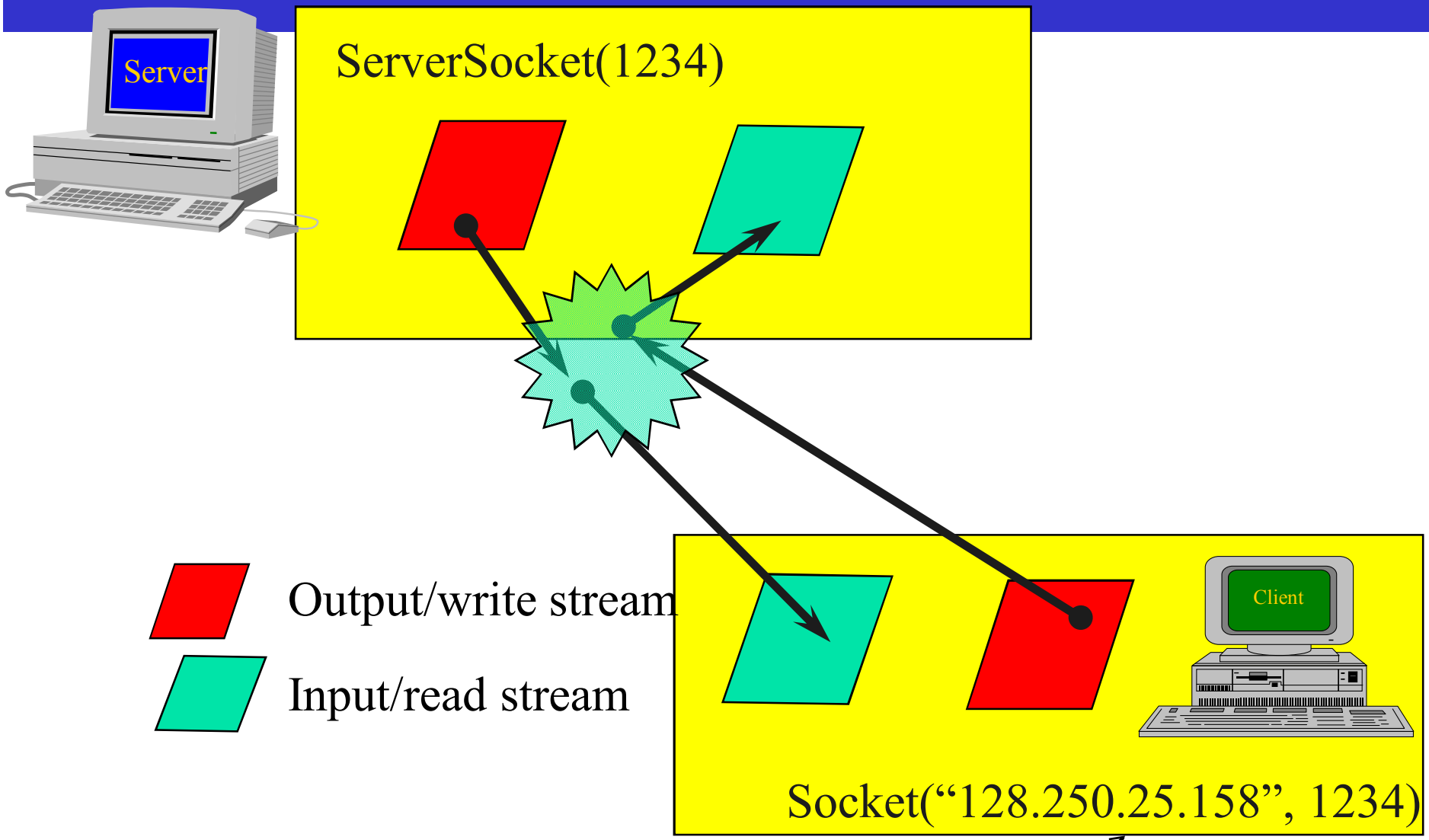
Example: client and server

- Program for server:
  - Creates a socket bound to server port(6789)
  - Repeatedly waits to receive a request msg
  - When received, it replies by sending back the same msg

UDP Sever: Devamlı olarak istek alıp client'a geri gönderir

```
import java.net.*;
import java.io.*;
public class UDPServer{
        public static void main(String args[]){
        DatagramSocket aSocket = null;
            try{
                aSocket = new DatagramSocket(6789);
                byte[] buffer = new byte[1000];
                while(true){
                    DatagramPacket request = new DatagramPacket(buffer, buffer.length);
                   aSocket.receive(request);
                   DatagramPacket reply = new DatagramPacket(request.getData(),
                        request.getLength(), request.getAddress(), request.getPort());
                   aSocket.send(reply);
                }
            }catch (SocketException e){System.out.println("Socket: " + e.getMessage());}
             catch (IOException e) {System.out.println("IO: " + e.getMessage());}
        finally {if(aSocket != null) aSocket.close();}
    }
}
```

TCP Stream için Java Soketleri



Server'ın Kodlaması

1. Open the Server Socket (Sunucu soketini aç):

```
ServerSocket server;
DataOutputStream os;
DataInputStream is;
server = new ServerSocket( PORT );
```

2. Wait for the Client Request (İstemcinin isteğini bekle):

```
Socket client = server.accept();
```

3. Client ile konuşmak için I/O akışları (stream) oluştur

```
is = new DataInputStream( client.getInputStream() );
os = new DataOutputStream( client.getOutputStream() );
```

4. Perform communication with client (İstemci ile iletişimi sağla)

```
Receive from client: String line = is.readLine();
Send to client: os.writeBytes("Hello\n");
```

5. Close sockets (Soketleri kapat): `client.close();`

multithreaded server için:

```
while(true) {
```

i. client isteği bekle (yukarıdaki 2. adım) ***ii.*** "client" soketini parametre olarak alan bir thread oluştur (thread, (3.) adımda olduğu gibi streamler oluşturur (4)'te belirtildiği gibi iletişim sağlar. ***iii.*** Servis sağlandığında thread'i kaldır.

```
}
```

Client'ın Kodlanması

1. Socket nesnesi oluştur:

```
client = new Socket( server, port_id );
```

2. Server ile iletişim kurmak için I/O stream'leri oluştur.

```
is = new DataInputStream(client.getInputStream() );
os = new DataOutputStream( client.getOutputStream() );
```

3. Server ile I/O veya iletişim sağla:
   - Server'dan veri al:

```
String line = is.readLine();
```

   - Server'a veri gönder:

```
os.writeBytes("Hello\n");
```

4. Bitince soketi kapat:

```
client.close();
```

Basit bir server (basitleştirilmiş kod)

```
// SimpleServer.java: basit bir sunucu programı
import java.net.*;
import java.io.*;
public class SimpleServer {
  public static void main(String args[]) throws IOException {
    // port 1234 üzerine servis kaydet
    ServerSocket s = new ServerSocket(1234);
    Socket s1=s.accept(); // Bekle ve bağlantı kabul et
    // Sokete iliştirilmiş bir iletişim stream'i al
    OutputStream s1out = s1.getOutputStream();
    DataOutputStream dos = new DataOutputStream (s1out);
    // string gönder!
    dos.writeUTF("Hi there");
    // Bağlantıyı kapat, server socket'ini değil
    dos.close();
    s1out.close();
    s1.close();
  }
}
```

Basit bir client (basitleştirilmiş kod)

```
// SimpleClient.java: basit bir istemci programı
import java.net.*;
import java.io.*;
public class SimpleClient {
  public static void main(String args[]) throws IOException {
    // Bir sunucuya, 1234 portunda bağlantı aç
    Socket s1 = new Socket("speedy.cs.pitt.edu",1234);
    // Soketten bir input file handle al ve input'u oku
    InputStream s1In = s1.getInputStream();
    DataInputStream dis = new DataInputStream(s1In);
    String st = new String (dis.readUTF());
    System.out.println(st);
    // Bitince bağlantıyı kapatıp çık
    dis.close();
    s1In.close();
    s1.close();
  }
}
```

Çalıştır

- speedy.cs.pitt.edu üzerindeki sunucuyu çalıştır
  - [user1@speedy] java SimpleServer &
- (speedy dahil) herhangi bir makinede istemciyi çalıştır :
  - [user2@speedy] java SimpleClient
    Hi there
- Server çalışmazken client'ı çalıştırırsanız:
  - [user2@speedy] sockets [1:147] java SimpleClient

```
Exception in thread "main" java.net.ConnectException: Connection refused
        at java.net.PlainSocketImpl.socketConnect(Native Method)
        at java.net.PlainSocketImpl.doConnect(PlainSocketImpl.java:320)
        at java.net.PlainSocketImpl.connectToAddress(PlainSocketImpl.java:133)
        at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:120)
        at java.net.Socket.<init>(Socket.java:273)
        at java.net.Socket.<init>(Socket.java:100)
        at SimpleClient.main(SimpleClient.java:6)
```

Socket Exceptions

```
try {
    Socket client = new Socket(host, port);
    handleConnection(client);
}
catch(UnknownHostException uhe) {
    System.out.println("Unknown host: " + host);
    uhe.printStackTrace();
}
catch(IOException ioe) {
System.out.println("IOException: " + ioe);
    ioe.printStackTrace();
}
```

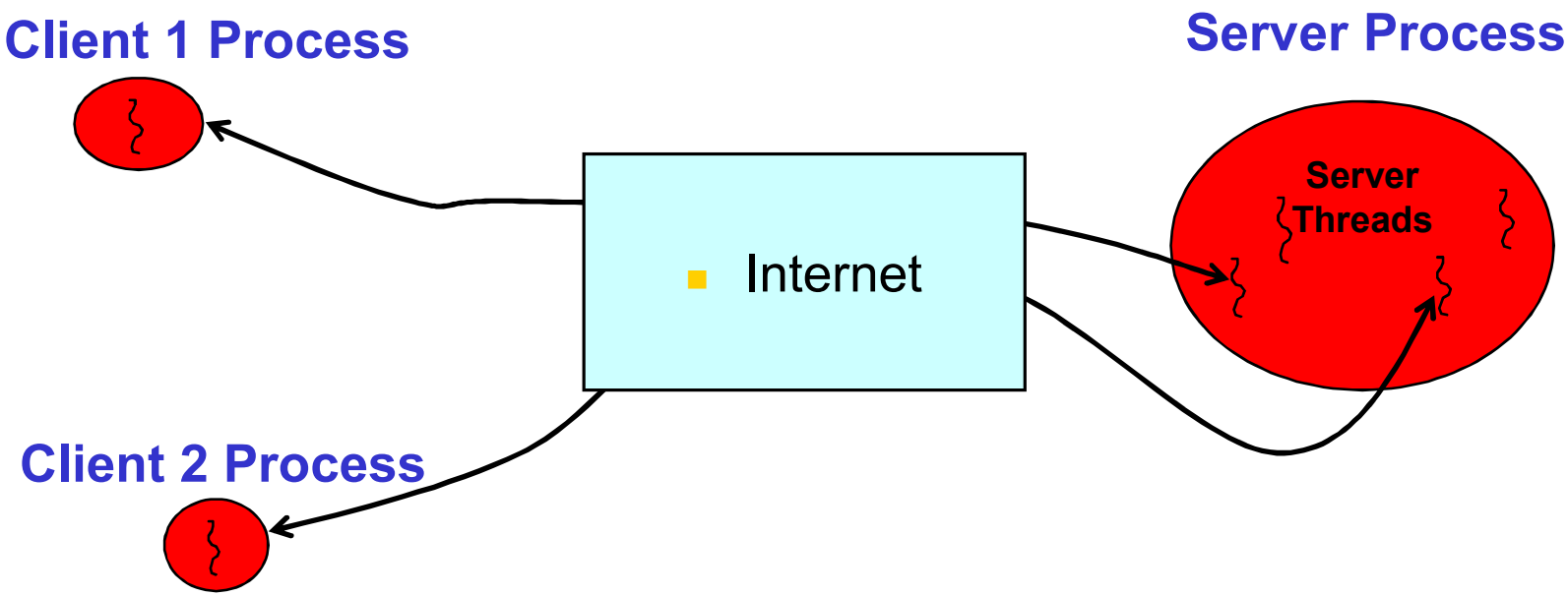
ServerSocket & Exceptions

- public **ServerSocket**(int port) throws IOException
  - Belirli bir port üzerinde server socket oluşturur.
  - port 0 ise, boş olan herhangi bir port üzerinde bir socket oluşturur. Bu soketin dinlediği (atanmış) portu belirlemek için **getLocalPort**()'u kullanabilirsiniz.
  - Gelen bağlantı belirtileri (bağlantı isteği) için en yüksek kuyruk (queue) uzunluğu 50 olarak ayarlanmıştır. Kuyruk doluyken bağlantı isteği ulaşırsa, bağlantı reddedilir.
- Throws:
  - IOException - soket açılırken I/O hatası olursa.
  - SecurityException - bir güvenlik yöneticisi varsa ve onun checkListen metodu (fonksiyonu) işleme izin vermezse.

Server Döngüsü: Devamlı çalışır

```
// SimpleServerLoop.java: a in tek bir thread içinde devamlı olarak çalışan basit bir server programı
import java.net.*;
import java.io.*;
public class SimpleServerLoop {
  public static void main(String args[]) throws IOException {
    // 1234 portu üzerinde servis kaydet
    ServerSocket s = new ServerSocket(1234);
    while(true)
    {
          Socket s1=s.accept(); // Bir bağlantı bekleyip kabul et
          // Sokete iliştirilmiş bir iletişim akışı al
          OutputStream s1out = s1.getOutputStream();
          DataOutputStream dos = new DataOutputStream (s1out);
          // string gönder!
          dos.writeUTF("Hi there");
          // Bağlantıyı kapat, server socket'i değil
          dos.close();
          s1out.close();
          s1.close();
    }
  }
}
```

Multithreaded Server: Birçok istemciye birden hizmet vermek için



Özet

- Java'da client/server uygulamaları programlamak eğlenceli ve zorludur.
- Java'da soket programlama, C gibi diğer dillerdekinden çok daha kolaydır.
- Anahtar kelimeler:
  - Clients, servers, TCP/IP, port number, sockets, Java sockets