

Appendix - Socket Programming

(reference only, not exam)

Textbook: Ch. 25

Outline

- ❖ Application using UDP
 - ❧ UDP server socket program
 - ❧ UDP client socket program


- ❖ Application using TCP
 - ❧ TCP server socket program
 - ❧ TCP client socket program

Demo using Java

- ❖ We follow the textbook to use Java as example.
 - ↪ Using other languages would be similar.
- ❖ In Java, an IP address is defined as an object, the instance of *InetAddress* class.
 - ↪ to deal with IP addresses and port numbers.
- ❖ For details,
 - ↪ <http://docs.oracle.com/javase/7/docs/api/java/net/InetAddress.html>

Iterative Programming Using UDP

- ❖ An **iterative server** can process one client request at a time;
 - ❖ it receives a ***request***, ***processes it***, and ***sends the response to the requestor*** before handling another request.
- ❖ When the server is handling the request from a client, the requests from other clients, and even other requests from the same client, *need to be queued* at the server site and wait for the server to be freed.
- ❖ The received and queued requests are handled in the first-in, first-out fashion.



Compare to
concurrent server

UDP communication

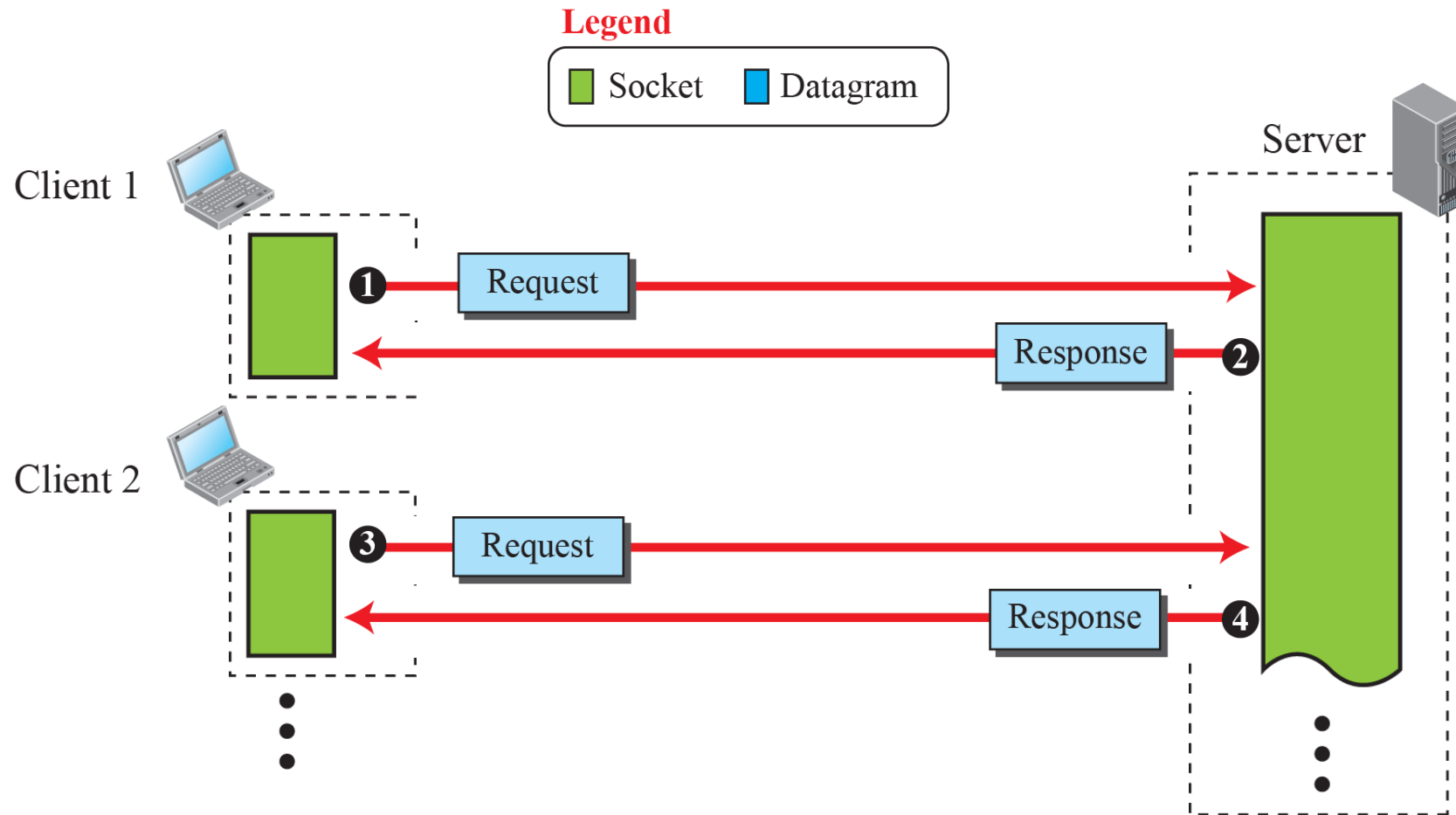


Figure 25.8: Sockets for UDP communication
SEHH2238 L12 Appendix

Get IP Addresses

```
GetIPAddress1.java ✕
1 // Table 25.6      Example 25.1
2
3 import java.net.*;
4 import java.io.*;
5 public class GetIPAddress1
6 {
7     public static void main (String [] args) throws IOException, UnknownHostException
8     {
9         InetAddress mysite = InetAddress.getByName ("polyu.edu.hk");
10        InetAddress local = InetAddress.getLocalHost ();
11        InetAddress addr = InetAddress.getByName ("202.126.57.188");
12
13        System.out.println (mysite);
14        System.out.println (local);
15
16        System.out.println (mysite.getHostAddress ());
17        System.out.println (local.getHostName ());
18        System.out.println (addr.getCanonicalHostName());
19
20    } // End of main
21
22 } // End of class
```

Problems @ Javadoc Declaration Console ✕

<terminated> GetIPAddress1 [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Feb 24, 2014 5:57:04 PM)

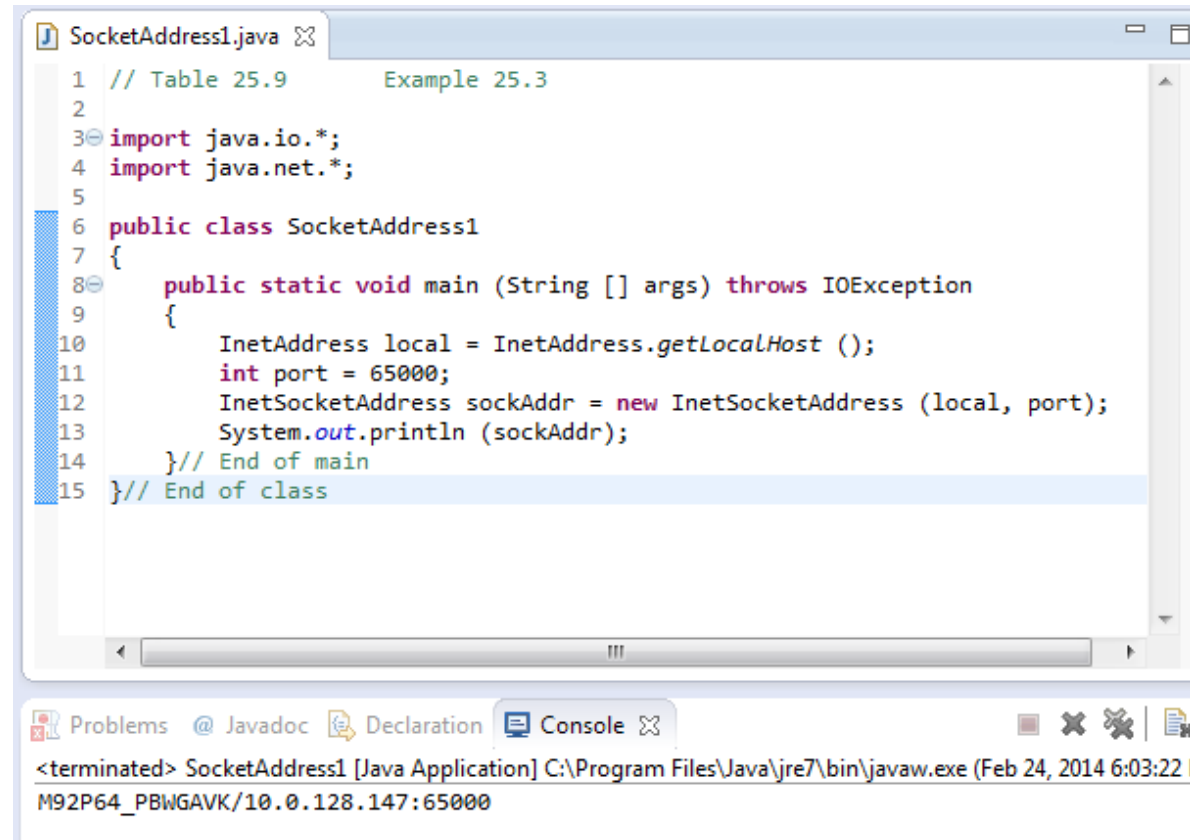
polyu.edu.hk/158.132.82.94
M92P64_PBWGAVK/10.0.128.147
158.132.82.94
M92P64_PBWGAVK
origin.www.tvb.com

First Example – Classes

- ❖ `getByName(String host)`
 - ↪ Determines the IP address of a host, given the host's name.
- ❖ `getLocalHost()`
 - ↪ Returns the address of the local host.
- ❖ `getHostAddress()`
 - ↪ Returns the IP address string in textual presentation.
- ❖ `public String getCanonicalHostName()`
 - ↪ Gets the fully qualified domain name for this IP address.

Socket Address

- ❖ This class implements an IP Socket Address (IP address + port number)



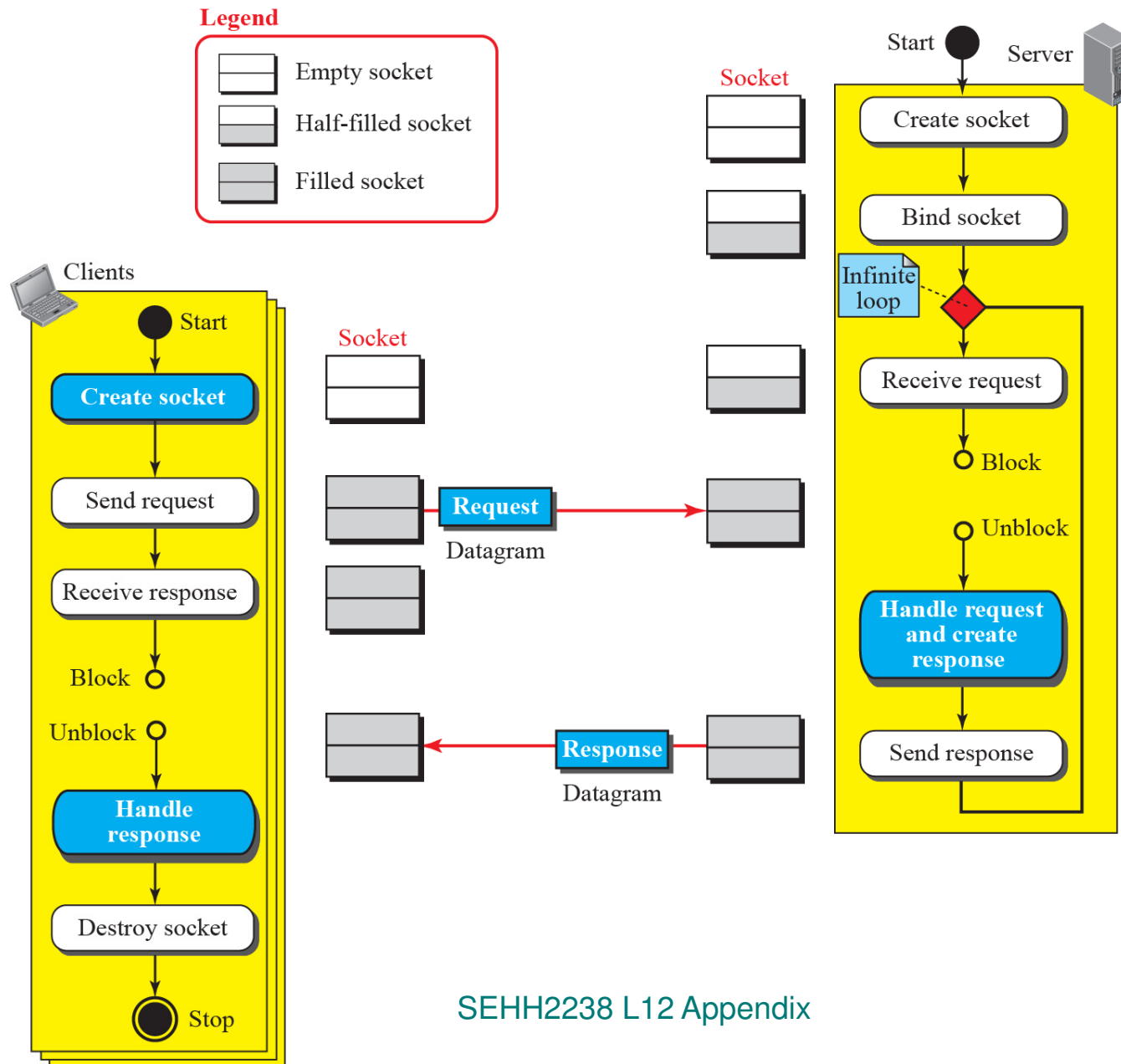
```
1 // Table 25.9      Example 25.3
2
3 import java.io.*;
4 import java.net.*;
5
6 public class SocketAddress1
7 {
8     public static void main (String [] args) throws IOException
9     {
10         InetAddress local = InetAddress.getLocalHost ();
11         int port = 65000;
12         InetSocketAddress sockAddr = new InetSocketAddress (local, port);
13         System.out.println (sockAddr);
14     } // End of main
15 } // End of class
```

Problems @ Javadoc Declaration Console

<terminated> SocketAddress1 [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Feb 24, 2014 6:03:22 PM)
M92P64_PBWGAVK/10.0.128.147:65000

🔗 <http://download.java.net/jdk7/archive/b123/docs/api/java/net/InetSocketAddress.html>

Figure 25.9: Flow diagram for iterative UDP communication



Classes for UDP

❖ DatagramSocket

- ❧ DatagramSocket Class is used to create sockets in the client and the server.
- ❧ It also provides methods to send a datagram, to receive a datagram, and to close the socket.

```
public class java.net.DatagramSocket extends java.lang.Object
```

// Constructors

```
public DatagramSocket ()
```

```
public DatagramSocket (int localPort)
```

```
public DatagramSocket (int localPort, InetAddress localAddr)
```

// Instance Methods

```
public void send (DatagramPacket sendPacket)
```

```
public void receive (DatagramPacket recvPacket)
```

```
public void close ()
```

Table 25.10: Some methods in DatagramSocket class

Classes for UDP

❖ DatagramPacket Class

∞ DatagramPacket class is used to create datagram packets.

```
public final class java.net.DatagramPacket extends java.lang.Object
```

// Constructors

```
public DatagramPacket (byte [] data, int length)
```

```
public DatagramPacket (byte [] data, int length, InetAddress remoteAddr, int remotePort)
```

// Instance Methods

```
public InetAddress getAddress ()
```

```
public int getPort ()
```

```
public byte [] getData ()
```

```
public int getLength ()
```

Table 25.11: Some methods in DatagramPacket class

Design of the UDP server

Server application program

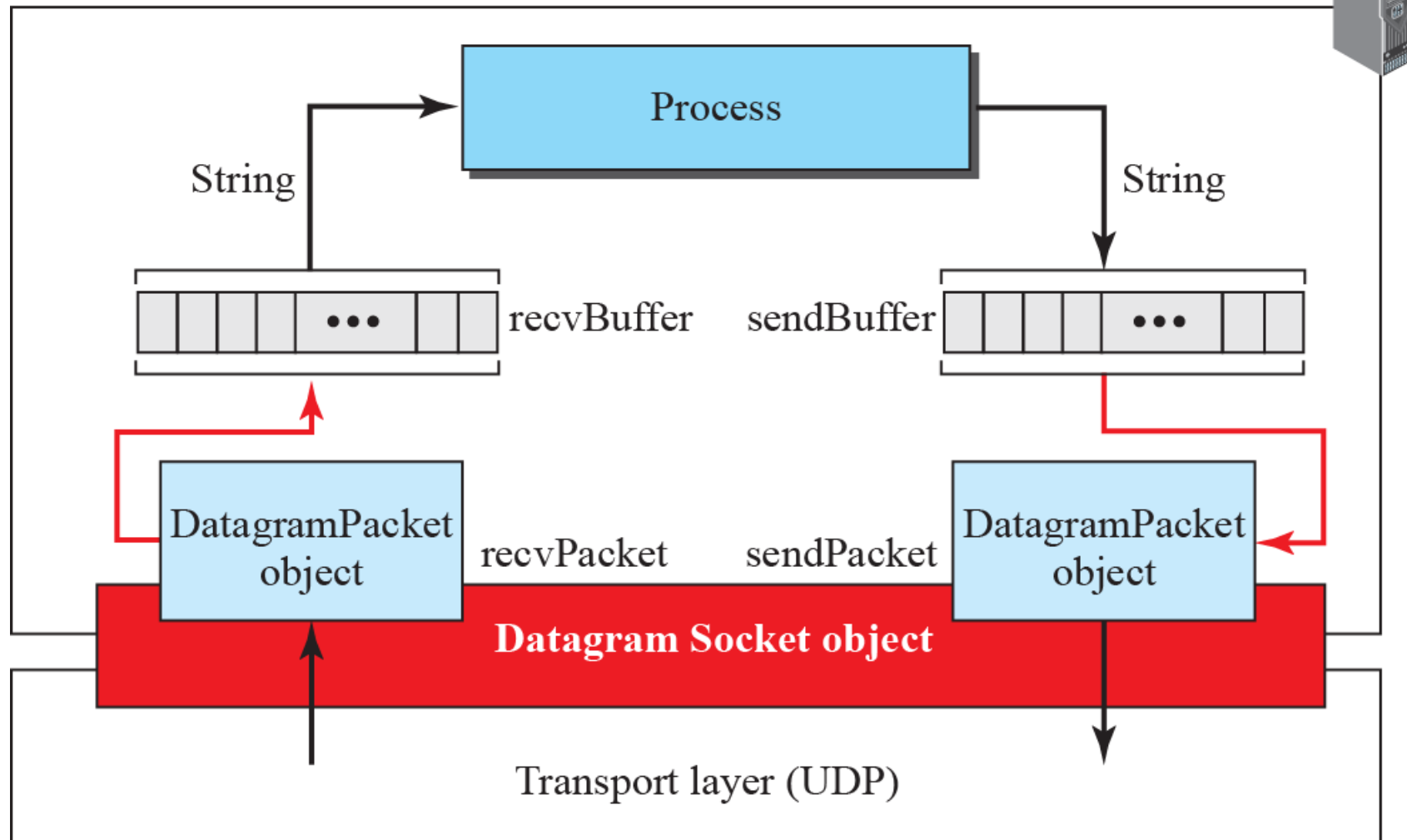


Figure 25.15:

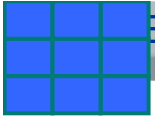


Table 25.12:

A simple UDP server program (Part I)

```
1 import java.net.*;
2 import java.io.*;
3
4 public class UDPServer
5 {
6     final int buffSize = ...;           // Add buffer size.
7     DatagramSocket sock;
8     String request;
9     String response;
10    InetAddress clientAddr;
11    int clientPort;
12
13    UDPServer (DatagramSocket s)
14    {
15        sock = s;
16    }
17
18    void getRequest ()
19    {
20        try
21        {
22            byte [] recvBuff = new byte [buffSize];
```

Allocate buffer

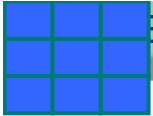


Table 25.12:

A simple UDP server program (Part II)

```
23 DatagramPacket recvPacket = new DatagramPacket (recvBuff, buffSize);
24 sock.receive (recvPacket);
25 recvBuff = recvPacket.getData ();
26 request = new String (recvBuff, 0, recvBuff.length);
27 clientAddr = recvPacket.getAddress ();
28 clientPort = recvPacket.getPort ();
29 }
30 catch (SocketException ex)
31 {
32     System.err.println ("SocketException in getRequest");
33 }
34 catch (IOException ex)
35 {
36     System.err.println ("IOException in getRequest");
37 }
38 }
39
40 void process ()
41 {
42     // Add code for processing the request and creating the response.
43 }
```

Wait for receiving

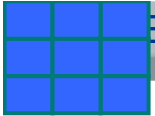


Table 25.12:

A simple UDP server program (Part III)

```
44
45 void sendResponse()
46 {
47     try
48     {
49         byte [] sendBuff = new byte [buffSize];
50         sendBuff = response.getBytes ();
51         DatagramPacket sandpaper = new DatagramPacket (sendBuff,
52                                                         sendBuff.length, clntAddr, clientPort);
53         sock.send(sendPacket);
54     }
55     catch (SocketException ex)
56     {
57         System.err.println ("SocketException in sendResponse");
58     }
59     catch (IOException ex)
60     {
61         System.err.println ("IOException in sendResponse");
62     }
63 }
64
```

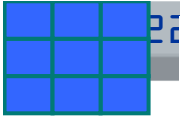


Table 25.12:

A simple UDP server program (Part IV)

```
65     public static void main (String [] args) throws IOException, SocketException
66     {
67         final int port = ...;           // Add server port number.
68         DatagramSocket sock = new DatagramSocket (port);
69         while (true)
70         {
71             UDPServer server = new UDPServer (sock);
72             server.getRequest ();
73             server.process ();
74             server.sendResponse ();
75         }
76     } // End of main
77 } // End of UDPServer class
```


UDP Server Program Structure

❖ **main** method

❧ It creates an instance of the DatagramSocket class using the defined port number.

❧ Then it runs an infinite loop.

❖ Each client is served in one iteration of the loop by

❧ Create a new instance of the UDPServer class and

❧ Call its three instance methods

❖ `getRequest()`

❖ `process()`

❖ `sendResponse()`

UDP Server Program Structure

❖ **getRequest** method

- ❧ Creates a receiver buffer (line 22)
- ❧ Creates the datagram packet and attaches it to buffer (line 23)
- ❧ Receives the datagram contents (line 24).
- ❧ Extracts the data part of the datagram and stores it in the buffer (line 25).
- ❧ Converts the bytes in the receiver buffer to the string request (line 26).
- ❧ Extracts the IP address of the client that sends the packet (line 27).
- ❧ Extracts the port number of the client that sends the request (line 28).

UDP Server Program Structure

❖ **sendRequest** method

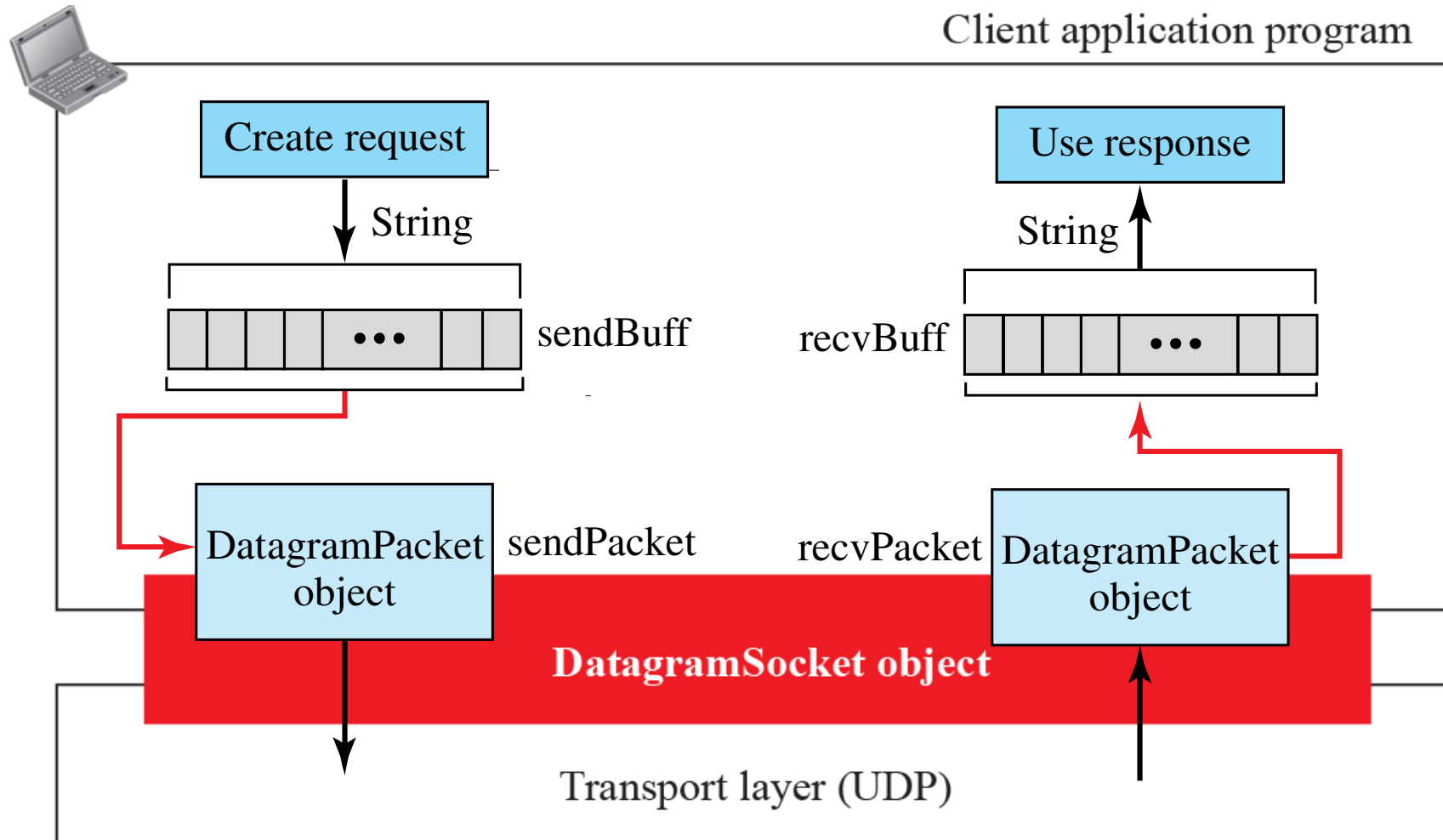
- ❧ Creates an empty send buffer (line 49).
- ❧ Changes the response string to bytes and stores them in the send buffer (line 50).
- ❧ Creates a new datagram and fills it with data in the buffer (line 51 and 52).
- ❧ Sends the datagram packet (line 53).

❖ **process** method

- ❧ To be worked out as application specific.

Figure 25.16:

Design of the UDP Client



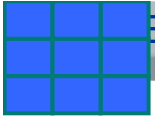


Table 25.13:

A simple UDP client program (Part I)

```
1 import java.net.*;
2 import java.io.*;
3
4 public class UDPClient
5 {
6     final int buffSize = ...;           // Add buffer size
7     DatagramSocket sock;
8     String request;
9     String response;
10    InetAddress servAddr;
11    int servPort;
12
13    UDPClient (DatagramSocket s, String sName, int sPort)
14                throws UnknownHostException
15    {
16        sock = s;
17        servAddr = InetAddress.getByName (sName);
18        servPort = sPort;
19    }
20
21    void makeRequest ()
22    {
23        // Code to create the request string to be added here.
24    }
25
```

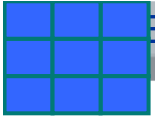


Table 25.13:

A simple UDP client program (Part II)

```
26 void sendRequest ()
27 {
28     try
29     {
30         byte [] sendBuff = new byte [buffSize];
31         sendBuff = request.getBytes ();
32         DatagramPacket sendPacket = new DatagramPacket (sendBuff,
33                                                         sendBuff.length, servAddr, servPort);
34         sock.send(sendPacket);
35     }
36     catch (SocketException ex)
37     {
38         System.err.println ("SocketException in getRequest");
39     }
40 }
41
42 void getResponse ()
43 {
44     try
45     {
46         byte [] recvBuff = new byte [buffSize];
47         DatagramPacket recvPacket = new DatagramPacket (recvBuff, buffSize);
48         sock.receive (recvPacket);
49         recvBuff = recvPacket.getData ();
50         response = new String (recvBuff, 0, recvBuff.length);
51     }
```

Wait for receiving

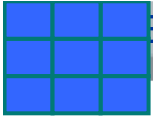


Table 25.13:

A simple UDP client program (Part III)

```
52      catch (SocketException ex)
53      {
54          System.err.println ("SocketException in getRequest");
55      }
56  }
57
58  void useResponse ()
59  {
60      // Code to use the response string needs to be added here.
61  }
62
63  void close ()
64  {
65      sock.close ();
66  }
67
```

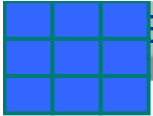


Table 25.13:

A simple UDP client program (Part IV)

```
68 public static void main (String [] args) throws IOException, SocketException
69 {
70     final int servPort = ...;           //Add server port number
71     final String servName = ...;       //Add server name
72     DatagramSocket sock = new DatagramSocket ();
73     UDPClient client = new UDPClient (sock, servName, servPort);
74     client.makeRequest ();
75     client.sendRequest ();
76     client.getResponse ();
77     client.useResponse ();
78     client.close ();
79 } // End of main
80 } // End of UDPClient class
```


UDP Client Program Structure

❖ **main** method

- ❧ Creates an instance of the DatagramSocket.
- ❧ Creates an instance of UDPClient class using the socket, server Name and server port number.
- ❧ Then it call its five instance methods
 - ❖ `makeRequest()`
 - ❖ `sendRequest()`
 - ❖ `getResponse()`
 - ❖ `useResponse()`
 - ❖ `close()`

UDP Client Program Structure

❖ **makeRequest** method

- ☞ To be worked out as application specific to create request string.

❖ **sendRequest** method

- ☞ Creates an empty send buffer (line 30).
- ☞ Fills up the send buffer with the request string created in the makeRequest method (line 31).
- ☞ Creates the datagram packet and attaches it to send buffer, server address, and the server port (line 32 and 33).
- ☞ Sends the packet using the send method defined in the DatagramSocket class (line 34).

UDP Client Program Structure

❖ **getResponse** method

- ❧ Creates an empty send buffer (line 46).
- ❧ Creates the receive datagram and attaches it to send buffer (line 47)
- ❧ Uses the receive method of the DatagramSocket to receive the response of the server and fills up the datagram with it (line 48).
- ❧ Extracts the data in the receive packet and stores it in the receive buffer (line 49).
- ❧ Creates the string response to be used by the `useResponse` method (line 50).

❖ **useResponse** method

- ❧ To be worked out as application specific.

❖ **close** method

- ❧ Close the socket.

Echo client/server

- ❖ The simplest example is to simulate the standard echo client/server.
- ❖ This program is used to check whether a server is alive.
- ❖ A short message is sent by the client.
- ❖ The message is exactly echoed back.
- ❖ Although the standard uses the well-known port 7 to simulate it, we use the port number 52007 for the server.

Client Program

1. In the client program, we set the server port to 52007, and the server name to the computer name or the computer address (x.y.z.t). We also change the makeRequest and useResponse methods as shown below:

```
void makeRequest ()  
{  
    request = "Hello";  
}
```

```
void useResponse ()  
{  
    System.out.println (response);  
}
```

Server Program

2. In the server, we set the server port to 52007.
We also replace the *process* methods in the server program to

```
void process ()  
{  
    response = request;  
}
```

Running

3. We let the server program run on one host and then run the client program on another host. We can use both on the same host if we run the server program in the background.

Simple date/time service - Client

1. In the client program, we set the server port to 40013 and set the server name to the computer name or the computer address (“x.y.z.t”). We also replace makeRequest and useResponse methods using the following code:

```
void makeRequest ()
{
    request = "Send me data and time please.";
}
```

```
void use Response ()
{
    System.out.println (response);
}
```


Simple date/time service - Server

2. In the server program (Table 25.12), we add one statement at the beginning of the program to be able to use the Calendar and the Date class (import java.util.*;). We set the server port to 40013. We also replace the process methods in the server program to

```
void process ()
{
    Date date = Calendar.getInstance ().getTime ();
    response = date.toString ();
}
```

Running

3. The process method uses the Calendar class to get the time (including the date) and then changes the date to a string to be stored in the response variable.
4. We let the server program run on one host and then run the client program on another host. We can use both on the same host if we run the server program in the background

Measuring the time

In this example, we need to use our simple client-server program to measure the time (in milliseconds) that it takes to send a message from the client to the server.

Client Program

1. In the client program, we add one statement at the beginning of the program to be able to use the Date class (import java.util.*;), we set the server port to 52007, and we set the server name to the computer name or the computer address (“x.y.z.t”). We also replace makeRequest and useResponse methods using the following code:

```
void makeRequest ()
{
    Date date = new Date ();
    long time = date.getTime ();
    request = String.valueOf (time);
}
```

```
void use Response ()
{
    Data date = new Date ();
    long now = date.getTime ();
    long elapsedTime = now - Long.parse(response));
    System.out.println ("Elapsed time = " + elapsedTime + " milliseconds.");
}
```

Server Program

2. In the server program, we set the server port to 52007. We also replace the process methods in the server program to

```
void process ()  
{  
    response = request;  
}
```

3. We let the server program run on one host and then run the client program on another host. We can use both on the same host if we run the server program in the background.

D. Iterative Programming Using TCP

- ❖ We are now ready to discuss network programming using the service of TCP, a connection-oriented service.
- ❖ Two classes designed for TCP:
 - ❧ ServerSocket Class
 - ❖ Create the listen sockets that are used for establishing communication in TCP (handshaking).
 - ❧ Socket Class
 - ❖ Used in TCP for data transfer.

Figure 25.11: Flow diagram for iterative TCP communication

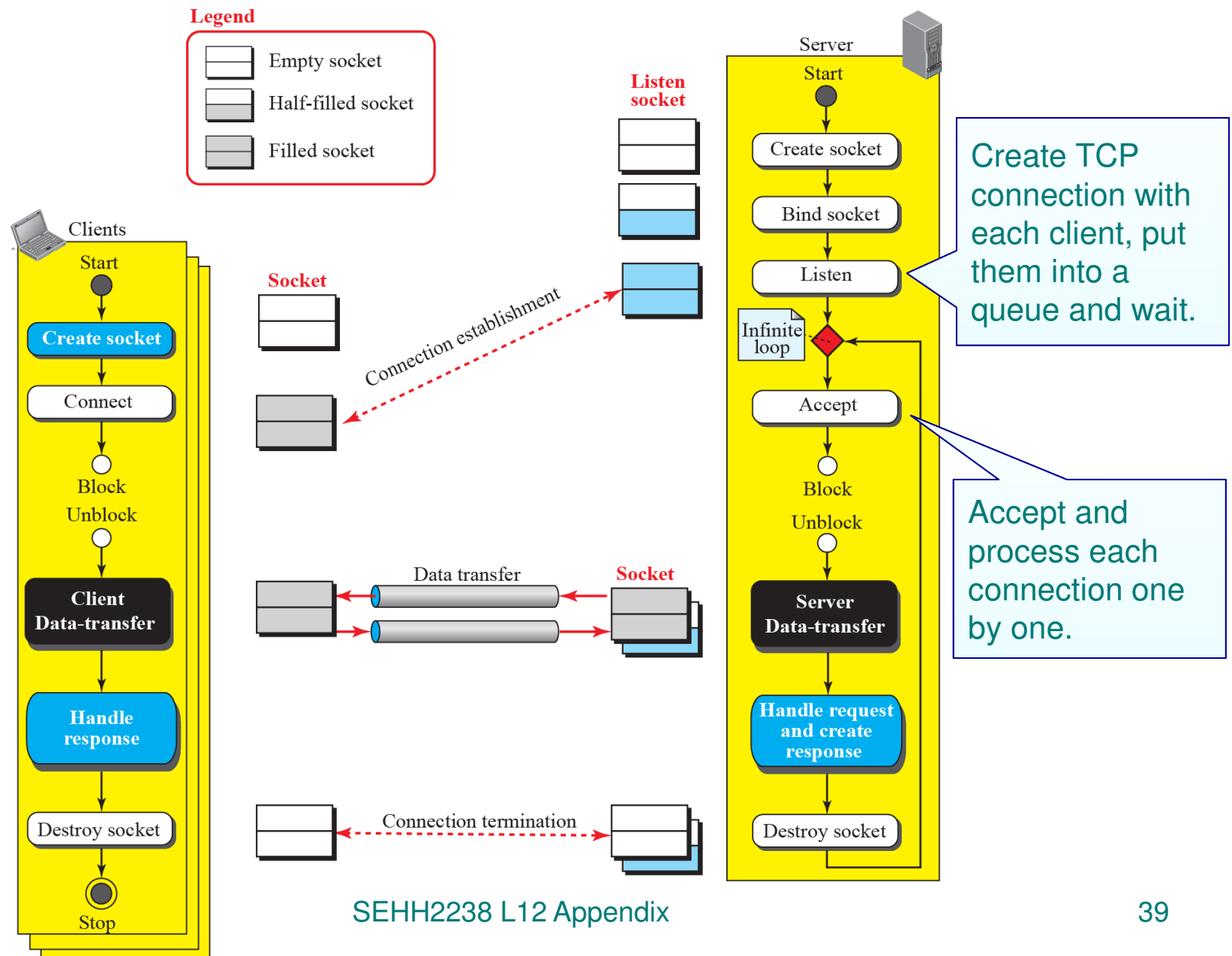
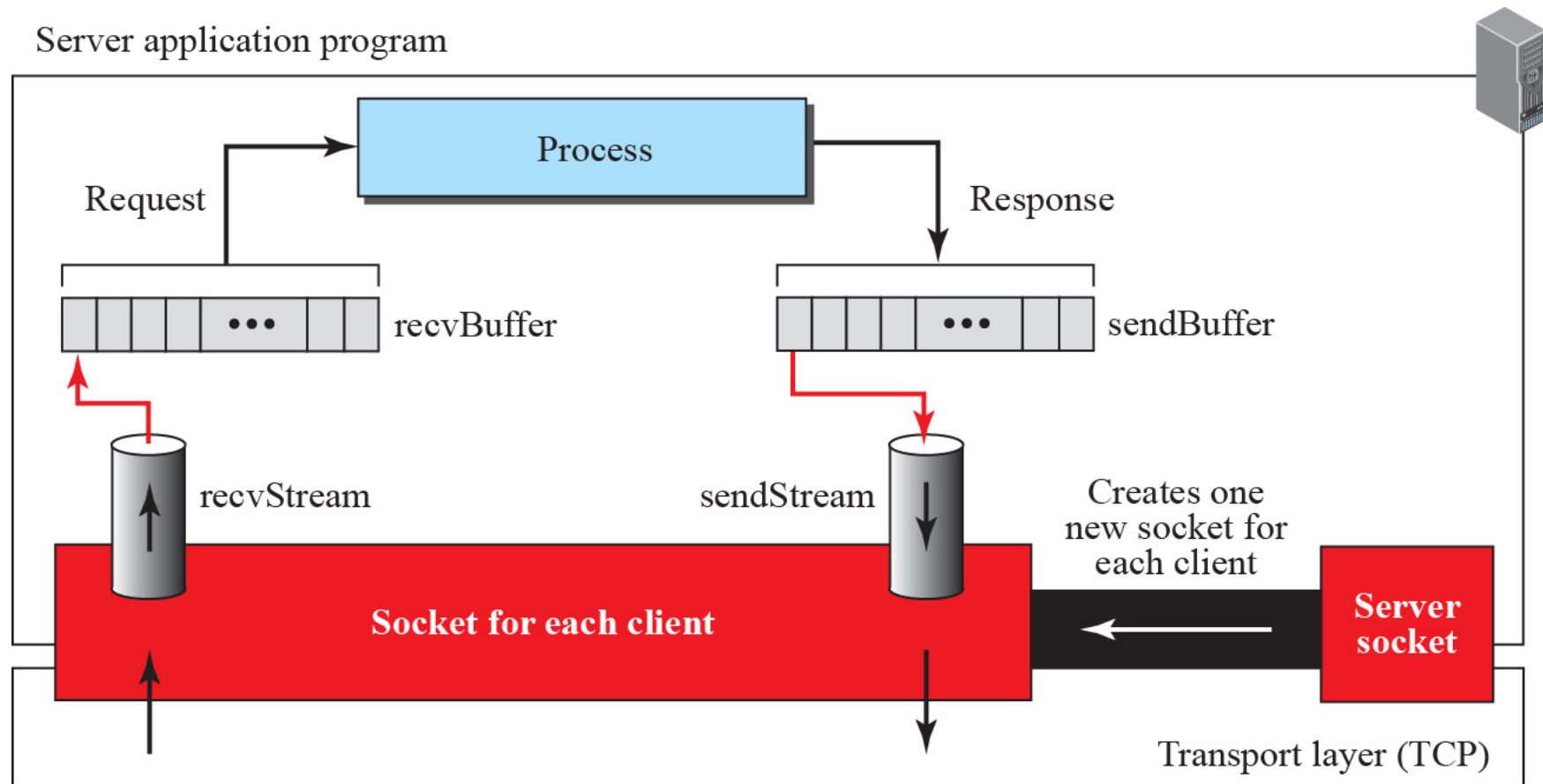


Figure 25.17:

Design of the TCP server for each client connection



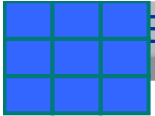


Table 25.16: A simple TCP server program (Part I)

```
1 import java.net.*;
2 import java.io.*;
3
4 public class TCPServer
5 {
6     Socket sock;
7     InputStream recvStream;
8     OutputStream sendStream;
9     String request;
10    String response;
11
12    TCPServer (Socket s) throws IOException, UnknownHostException
13    {
14        sock = s;
15        recvStream = sock.getInputStream ();
16        sendStream = sock.getOutputStream ();
17    }
18
19    void getRequest ()
20    {
21        try
22        {
```

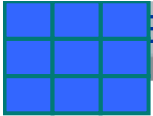


Table 25.16: A simple TCP server program (Part II)

```
23         int dataSize;
24         while ((dataSize = recvStream.available ()) == 0);
25         byte [] recvBuff = new byte [dataSize];
26         recvStream.read (recvBuff, 0, dataSize);
27         request = new String (recvBuff, 0, dataSize);
28     }
29     catch (IOException ex)
30     {
31         System.err.println ("IOException in getRequest");
32     }
33 }
34
35 void process()
36 {
37     // Add code to process the request string and create response string.
38 }
39
40 void sendResponse ()
41 {
42     try
43     {
44         byte [] sendBuff = new byte [response.length ()];
45         sendBuff = response.getBytes ();
46         sendStream.write (sendBuff, 0, sendBuff.length);
47     }
```

Wait until there is
data received

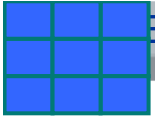


Table 25.16: A simple TCP server program (Part III)

```
48         catch (IOException ex)
49         {
50             System.err.println ("IOException in sendResponse");
51         }
52     }
53
54     void close ()
55     {
56         try
57         {
58             recvStream.close ();
59             sendStream.close ();
60             sock.close ();
61         }
62         catch (IOException ex)
63         {
```

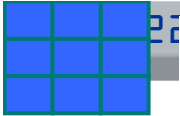
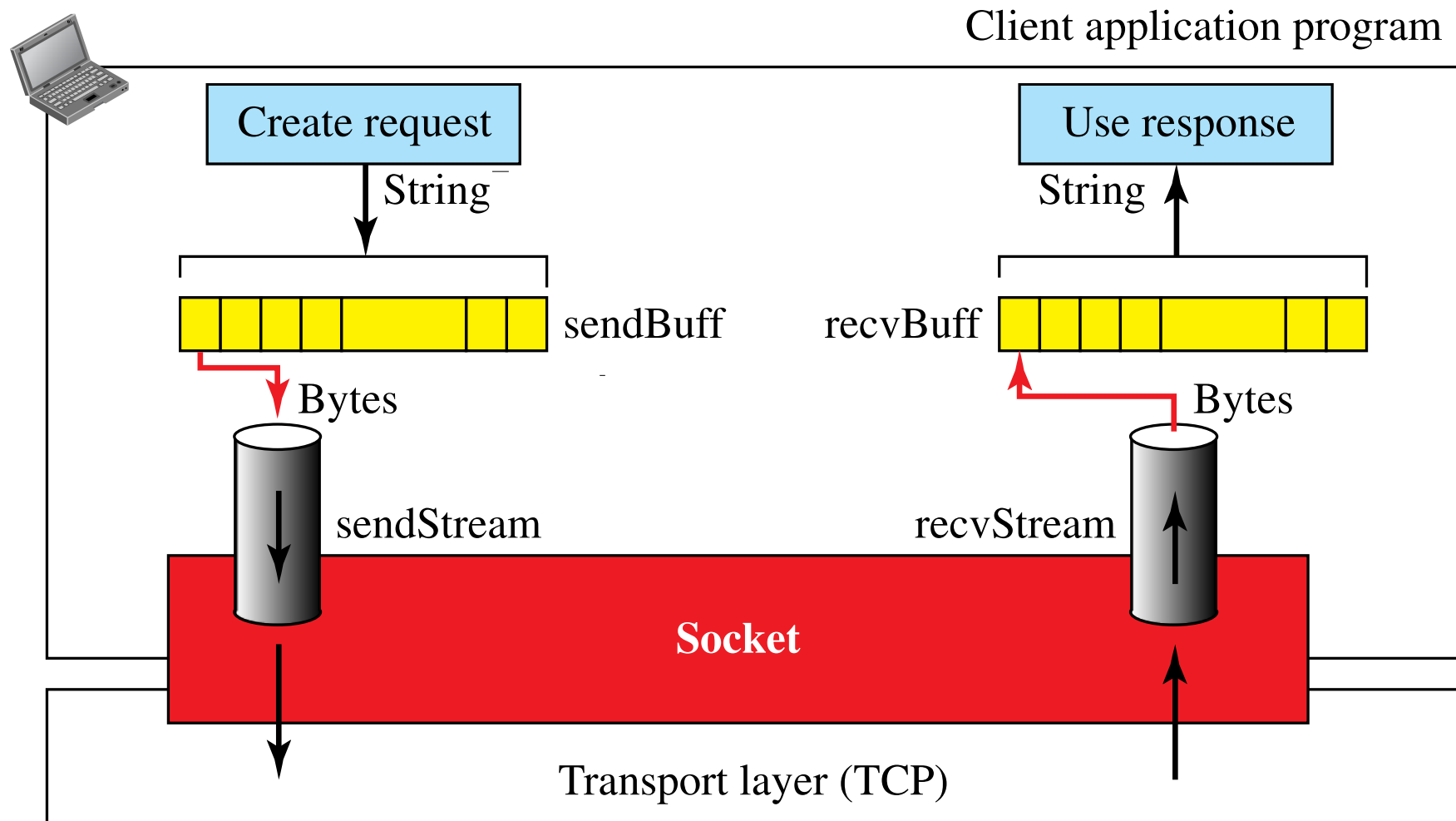


Table 25.16: A simple TCP server program (Part IV)

```
64         System.err.println ("IOException in close");
65     }
66 }
67
68 public static void main (String [] args) throws IOException
69 {
70     final int port = ...;           // Provide port number
71     ServerSocket listenSock = new ServerSocket (port);
72     while (true)
73     {
74         TCPServer server = new TCPServer (listenSock.accept ());
75         server.getRequest ();
76         server.process ();
77         server.sendResponse ();
78         server.close ();
79     }
80 } // End of main
81 // End of TCPServer class
```

Figure 25.18: Design of the TCP client



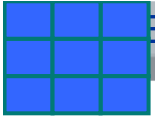


Table 25.17: A simple TCP client program (Part I)

```
1 import java.net.*;
2 import java.io.*;
3
4 public class TCPClient
5 {
6     Socket sock;
7     OutputStream sendStream;
8     InputStream recvStream;
9     String request;
10    String response;
11
12    TCPClient (String server, int port) throws IOException, UnknownHostException
13    {
14        sock = new Socket (server, port);
15        sendStream = sock.getOutputStream ();
16        recvStream = sock.getInputStream ();
17    }
18
19    void makeRequest ()
20    {
21        // Add code to make the request string here.
22    }
```

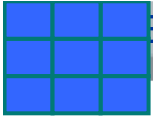


Table 25.17: A simple TCP client program (Part II)

```
23
24 void sendRequest ()
25 {
26     try
27     {
28         byte [] sendBuff = new byte [request.length ()];
29         sendBuff = request.getBytes ();
30         sendStream.write (sendBuff, 0, sendBuff.length);
31     }
32     catch (IOException ex)
33     {
34         System.err.println ("IOException in sendRequest");
35     }
36 }
37
38 void getResponse ()
39 {
40     try
41     {
42         int dataSize;
```

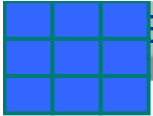


Table 25.17: A simple TCP client program (Part III)

```
43         while ((dataSize = recvStream.available ()) == 0);
44         byte [] recvBuff = new byte [dataSize];
45         recvStream.read (recvBuff, 0, dataSize);
46         response = new String (recvBuff, 0, dataSize);
47     }
48     catch (IOException ex)
49     {
50         System.err.println ("IOException in getResponse");
51     }
52 }
53
54 void useResponse ()
55 {
56     // Add code to use the response string here.
57 }
58
59 void close ()
60 {
61     try
62     {
63         sendStream.close ();
64         recvStream.close ();
65         sock.close ();
66     }
```

Wait until there is
data received

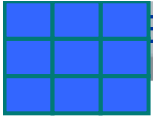


Table 25.17: A simple TCP client program (Part IV)

```
67         catch (IOException ex)
68         {
69             System.err.println ("IOException in close");
70         }
71     }
72
73     public static void main (String [] args) throws IOException
74     {
75         final int servPort = ...;           // Provide server port
76         final String servName = "...";      // Provide server name
77         TCPClient client = new TCPClient (servName, servPort);
78         client.makeRequest ();
79         client.sendRequest ();
80         client.getResponse ();
81         client.useResponse ();
82         client.close ();
83     } // End of main
84 } // End of TCPClient class
```

Summary

❖ Socket Address

- ↪ IP Address + Port Number

❖ Socket Programming

- ↪ Connect applications to the TCP/IP Protocols suite

- ↪ Client side vs. Server side, UDP vs. TCP

❖ Revision Quiz

- ↪ http://highered.mheducation.com/sites/0073376221/student_view0/chapter25/quizzes.html