

Java Networking

Overview

- Java language has support for host Internet Protocol
- Internet Protocol is a family that includes:
 - Routing between networks
 - Reliable, connection-oriented communications via TCP (Transport Control Protocol)
 - Unreliable, datagram communications via UDP (User Datagram Protocol)
- Some applications are:
 - email (SMTP)
 - file transfer (FTP)
 - Web (HTTP)
 - BitTorrent

Client-Server Networking Model

- Networking will be examined using the client-server model, where a *server* waits for requests from a *client* and the *client* makes requests to the *server*.
- Web browsers and HTTP servers fit this model where the browser is the client, requesting HTML and other types of files from a Web server.
- Although other protocols exist, our examination will be restricted to the commonly used TCP/IP, a connection-oriented protocol that guarantees reliable delivery of data.

Client-Server Responsibilities

- *Server*

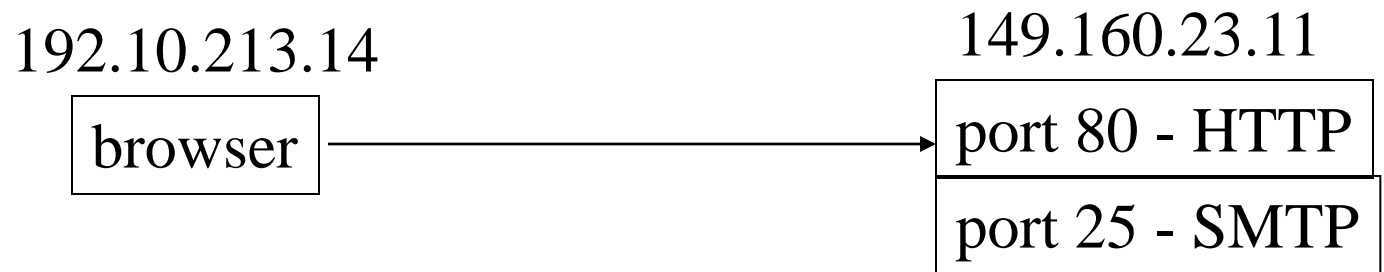
1. Define a single *port* number that clients can communicate through to the server.
2. Wait for a client to connect to that *port*. Since multiple clients can communicate through a single *port*, a bi-directional *socket* is opened to each client.
3. Send and receive bytes through the *socket*.

- *Client*

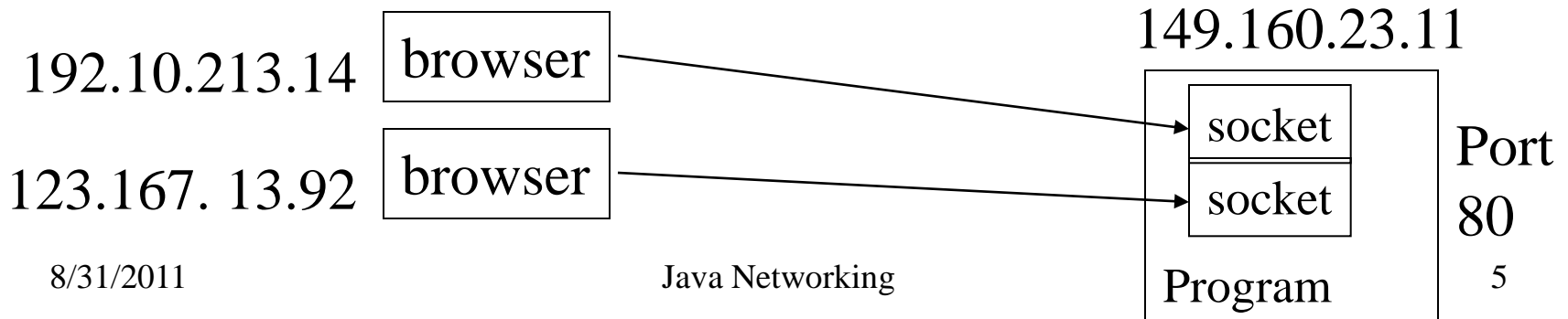
1. Open a connection to a server *port* through a bi-directional *socket*.
2. Send and receive bytes through the *socket*.

Internet Addressing

- Each host (usually meaning a computer) has a unique address.
- IP (*Internet Protocol*) address is a four octet such as 149.160.23.11
- IP address corresponds to the *name* such as *ps100-13.ius.indiana.edu*.
- Connection made through a unique *port* on a computer at an address.

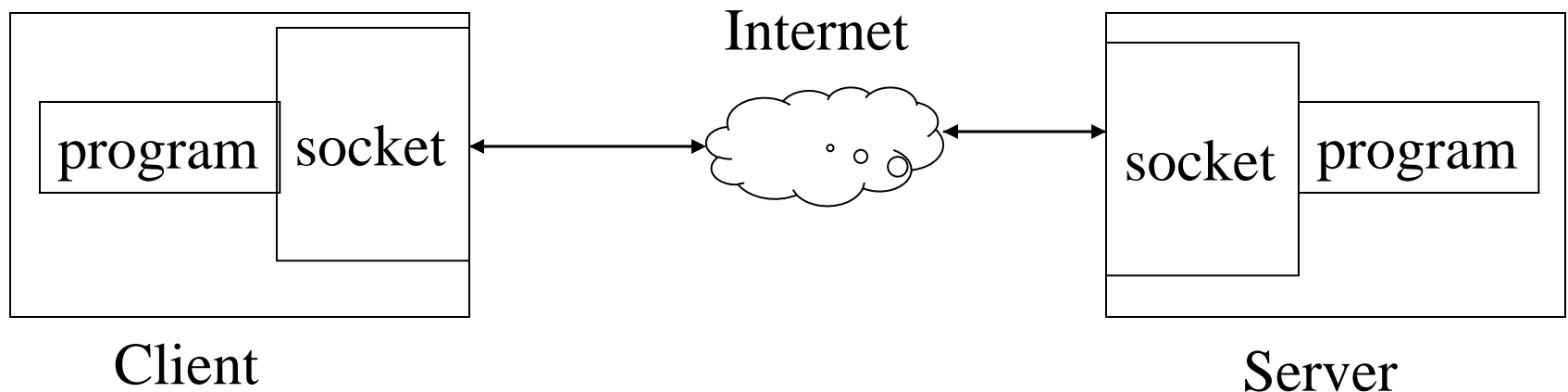


- Since several connections may occur simultaneously at each port/address, each connection has a unique (on that computer) *socket*.



Communication Model

- A single *client* and *server* program connected through a *bi-directional* channel such as a LAN
- *sockets* serve to connect a **program** to the network.
- Programs on different computers communicate through their socket.
- Once established, using the connection is much like reading and writing a file.



“Hello World” Client/Server

1. *Server* listens for connection on port 888.
2. *Client* connects on port 888.
3. *Server* writes out “Hello World” to connection.
4. *Client* reads in “Hello World” from connection.



Client

Server

“Hello World” Client/Server

Server

```
A. import java.net.*;
B. import java.io.*;
C. public class Server {
D.     public static void main(String args[]) throws Exception {

E.         ServerSocket connection = new ServerSocket( 888 );
F.         Socket s = connection.accept();           // connection wait
G.         PrintStream out = new PrintStream(s.getOutputStream( ) );

H.         out.println("Hello World");
I.     }
J. }
```

Client

```
a. import java.net.*;
b. import java.io.*;
c. public class Client {
d.     public static void main(String args[]) throws Exception {

e.         Socket s = new Socket("localhost", 888 );    // Connect
f.         BufferedReader in = new BufferedReader(
g.             new InputStreamReader( s.getInputStream( ) ) );

h.         System.out.println( in.readLine() );
i.     }
j. }
```

1. *(F)* *Server* listens for connection on port 888.
2. *(e)* *Client* connects on port 888.
3. *(H)* *Server* writes out “Hello World” to connection.
4. *(h)* *Client* reads in “Hello World” from connection.

Exercise 1 - List the lines in the order of execution.

Non-stop “Hello World” Client/Server

1. *Server* listens for connection on port 888.
2. *Client* connects on port 888.
3. *Server* writes out “Hello World” to connection.
4. *Client* reads in “Hello World” from connection.
5. ***Server* closes connection, go to 1.**

2. *Client* connects on port 888.
4. *Client* reads in “Hello World” from connection.

Client

“Hello World”

1. *Server* listens for connection on port 888.
3. *Server* writes out “Hello World” to connection.
5. ***Server* closes connection, go to 1.**

Server

Server

```
A. import java.net.*;
B. import java.io.*;
C. public class Server {
D.     public static void main(String args[]) throws Exception {
E.         ServerSocket connection = new ServerSocket( 888 );
F.         while(true) {
G.             Socket s = connection.accept();           // connection wait
H.             PrintStream out=new PrintStream(s.getOutputStream());
I.             out.println("Hello World");
J.         }
K.     }
L. }
```

Client

```
a. import java.net.*;
b. import java.io.*;
c. public class Client {
d.     public static void main(String args[]) throws Exception {
e.         Socket s = new Socket("localhost", 888 );    // connect
f.         BufferedReader in = new BufferedReader(
g.             new InputStreamReader( s.getInputStream( ) ) );
h.         System.out.println( in.readLine() );
i.     }
j. }
```

Non-stop “Hello World” Client/Server

1. *(G)* Server listens for connection on port 888.
3. *(I)* Server writes out “Hello World” to connection.
5. *Server* closes connection, go to 1.
2. *(e)* *Client* connects on port 888.
4. *(h)* *Client* reads in “Hello World” from connection.

Exercise 2 - List the lines in the order of execution.

Web Client/Server

Client program steps:

1. Connects to server *port 80* at server IP address.
2. Sends the server message:
GET / HTTP/1.0
meaning to get the default homepage and that the client understands HTTP 1.0 protocol
3. Browser reads HTML from server then renders HTML on display.

Server program steps:

1. Waits for connection on *port 80*
2. Inputs messages such as:
GET / HTTP/1.0
meaning to get the default homepage and that the client understands HTTP 1.0 protocol
3. Server sends homepage written in HTML.
4. Closes the connection.

Web Client

Client program steps:

1. Connects to server at *port 80* and IP address “www.ius.edu”.

```
Socket s = new Socket("www.ius.edu", 80 );
```

2. Sends server the message:

```
GET / HTTP/1.0
```

meaning to get the default homepage and that the client understands HTTP 1.0 protocol

```
out.print("GET / HTTP/1.0\n\n");
```

3. Client reads HTML returned from Server.

```
while((from=in.readLine()) != null )
```

```
System.out.println(from);
```

Java Web Client

```
1. import java.net.*;
2. import java.io.*;
3. public class simpleClient {
4.     public static void main(String args[]) throws Exception {
5.         Socket s = new Socket("www.ius.edu", 80 );
6.         BufferedReader in = new BufferedReader(
7.             new InputStreamReader( s.getInputStream( ) ) );
8.         PrintStream out = new PrintStream(s.getOutputStream());
9.         out.print("GET / HTTP/1.0\n\n");
10.        String from;
11.        while((from=in.readLine()) != null)
12.            System.out.println(from);
13.        System.out.println("Server closed connection");
14.    }
15. }
```

Web Server

Server program steps:

1. Wait for connection on *port 80* and creates socket.
ServerSocket connection = new ServerSocket(80);
Socket s = connection.accept();
2. Create input/output stream through socket
BufferedReader in = new BufferedReader(
 new InputStreamReader(s.getInputStream()));
PrintStream out = new PrintStream(s.getOutputStream());
3. Input messages from client such as:
 GET / HTTP/1.0
 meaning to get the default homepage and that the client understands HTTP 1.0 protocol. Our server just prints input.
 while(!(from=in.readLine()).equals(""))
 System.out.print(from);
4. Send HTML file. Ours is “Hello World”.
out.print("<html><H1>Hello World</H1></html>");
5. Close the connection.
s.close();

Java Web Server

```
1. import java.net.*;
2. import java.io.*;
3. public class simpleServer {
4.     public static void main(String args[]) throws Exception {
5.         ServerSocket connection = new ServerSocket( 80 );
6.         Socket s = connection.accept();
7.         BufferedReader in = new BufferedReader(
8.             new InputStreamReader( s.getInputStream( ) ));
9.         PrintStream out = new PrintStream(s.getOutputStream());
10.        String from;
11.        while( !( from=in.readLine() ).equals("") )
12.            System.out.print( from );
13.        out.print("<html><H1>Hello World</H1></html>");
14.        s.close();
15.    }
16.}
```

Server

```
A. public class simpleServer {  
B.     public static void main(String args[]) throws Exception {  
C.         ServerSocket connection = new ServerSocket( 80 );  
D.         Socket s = connection.accept();           // Connection wait  
E.         BufferedReader in = new BufferedReader(  
F.             new InputStreamReader( s.getInputStream( ) ) );  
G.         PrintStream out = new PrintStream(s.getOutputStream( ) );  
H.         String from;  
I.         while( !( from=in.readLine() ).equals("") ) System.out.print( from );  
J.         out.print("<html><H1>Hello World</H1></html>");  
K.         s.close();
```

```
a. public class simpleClient {  
b.     public static void main(String args[]) throws Exception {  
c.         Socket s = new Socket( "www.ius.edu" 80 );    // Connect  
d.         BufferedReader in = new BufferedReader(  
e.             new InputStreamReader( s.getInputStream( ) ) );  
f.         PrintStream out = new PrintStream(s.getOutputStream());  
g.         out.println("GET / HTTP/1.0\n");  
h.         String from;  
i.         while((from=in.readLine()) != null) System.out.println(from);  
j.         System.out.println("Server closed connection");
```

Client

Exercise 3 -
List the
lines in the
order of
execution.

Server

1. Copy *simpleServer.java* from above example.
2. If at IUS, at DOS prompt: `v:\common\user\c311\forJava`
3. `javac -classpath . simpleServer.java`
`java -cp . simpleServer`
4. In browser enter URL of *localhost* and click *Reload* OR
at DOS prompt: `java -cp . simpleClient`

Client

1. Copy *simpleClient.java* from above example.
2. If at IUS, at DOS prompt: `v:\common\user\c311\forJava`
3. `javac -classpath . simpleClient.java`
`java -cp . simpleClient`

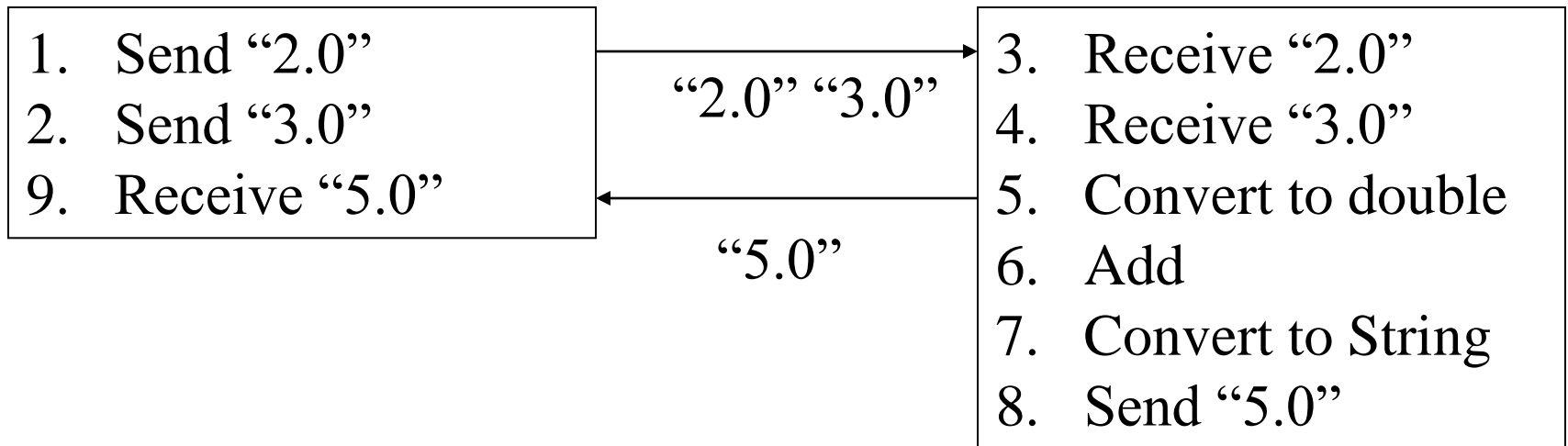
Object Serialization

- Client/Server communicate by data protocol
- HTTP used for Web browsers and servers
- Overhead for passing and parsing character stream:
<html><H1>Hello World</H1></html>
- One solution is to pass a complete object over the wire rather than a stream of characters
- Requires same object representation on client and server.
- OK when Java client/server.

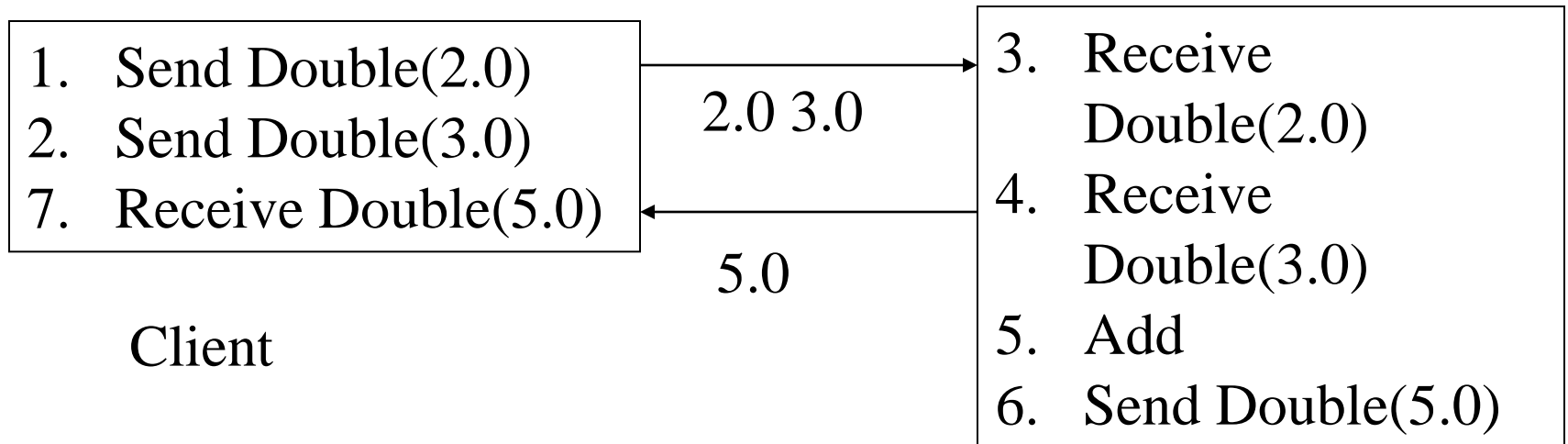
- `out.writeObject(object)` serializes *object* and writes to *out* stream
- `in.readObject()` reads *in* stream and de-serializes to an *Object*. Must then be cast to proper class.

Text versus Object passing (in theory)

Text



Object



Client

Server

Text passing Addition Client/Server

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

```
A. public class Server {  
B.     public static void main(String args[]) throws Exception {  
C.         ServerSocket connection = new ServerSocket( 888 );  
D.         Socket s = connection.accept();  
E.         PrintStream out = new PrintStream( s.getOutputStream( ) );  
F.         BufferedReader in = new BufferedReader( new InputStreamReader( s.getInputStream( ) ) );  
G.         double x = (new Double(in.readLine())).doubleValue();  
H.         double y = (new Double(in.readLine())).doubleValue();  
I.         out.println(x+y);  
J.     }  
K. }
```

Server

Exercise 4

1. What is sent at line f?
2. What is received at line h?

```
a. public class Client {  
b.     public static void main(String args[]) throws Exception {  
c.         Socket s = new Socket("localhost", 888 );  
d.         BufferedReader in = new BufferedReader( new InputStreamReader( s.getInputStream( ) ) );  
e.         PrintStream out = new PrintStream(s.getOutputStream( ) );  
f.         out.println(2.0);  
g.         out.println(3.0);  
h.         System.out.println( in.readLine() ); // Print x+y  
i.     }  
j. }
```

Client

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

Object passing Addition Client/Server

```
public class SerializeServer {
    public static void main(String args[]) throws Exception {
        ServerSocket connection = new ServerSocket( 888 );
        Socket s = connection.accept();
        ObjectOutputStream out = new ObjectOutputStream(s.getOutputStream( ));
        ObjectInputStream in = new ObjectInputStream(s.getInputStream( ));
        double x = ((Double)in.readObject()).doubleValue();
        double y = ((Double)in.readObject()).doubleValue();
        out.writeObject(new Double(x+y));
    }
}
```

Server

Exercise 5

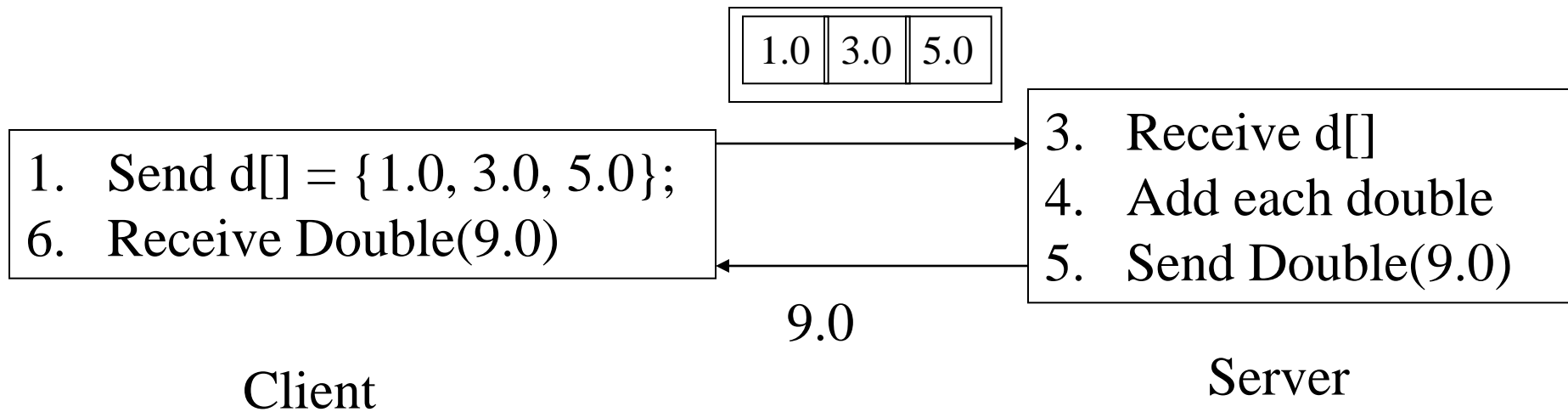
1. What is sent at line f?
2. What is received at line h?

```
public class SerializeClient {
    public static void main(String args[]) throws Exception {
        Socket s = new Socket("localhost", 888 );
        ObjectOutputStream out = new ObjectOutputStream(s.getOutputStream( ));
        ObjectInputStream in = new ObjectInputStream(s.getInputStream( ));
        out.writeObject(new Double(2.0));
        out.writeObject(new Double(3.0));

        System.out.println( (Double)in.readObject() ); // Print x + y
    }
}
```

Client

Array Object passing (in practice)



Array Object passing Addition Client/Server

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

```
A. public class SerializeServer {  
B.     public static void main(String args[]) throws Exception {  
C.         ServerSocket connection = new ServerSocket( 888 );  
D.         Socket s = connection.accept();  
E.         ObjectOutputStream out = new ObjectOutputStream(s.getOutputStream( ));  
F.         ObjectInputStream in = new ObjectInputStream(s.getInputStream( ) );  
G.         double sum=0.0 , d[]= (double []) in.readObject();  
H.         for(int i=0; i<d.length; i++) sum = sum + d[i];  
I.         out.writeObject(new Double(sum));  
J.     }  
K. }
```

Server

Exercise 6

1. What is sent at line g?
2. What is received at line G?

```
a. public class SerializeClient {  
b.     public static void main(String args[]) throws Exception {  
c.         Socket s = new Socket("localhost", 888 );  
d.         ObjectOutputStream out = new ObjectOutputStream(s.getOutputStream( ));  
e.         ObjectInputStream in = new ObjectInputStream(s.getInputStream( ) );  
f.         double d[] = { 1.0, 3.0, 5.0 };  
g.         out.writeObject( d );  
h.         System.out.println( (Double)in.readObject() ); // Print 1.0 + 3.0 + 5.0  
i.     }  
j. }
```

Client

Networking Summary

- Networking not part of language, implemented in package *java.net*
- Transport Control Protocol (TCP) and User Datagram Protocol (UDP) supported.
- Secure Socket Layer (SSL) supported.
- Threaded objects cannot currently be serialized