ڤوليتيكنيك بروني
POLITEKNIK BRUNEI

NS4307
Network Programming

Java Socket

Review

- The Internet Protocol is a low-level protocol for delivering data from one computer to another across the Internet in packets.

- Two higher-level protocols used in conjunction with the IP are the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP).

Review (cont.)

- TCP enables two hosts to establish a connection and exchange streams of data.
- TCP guarantees delivery of data and also guarantees that packets will be delivered in the same order in which they were sent.
- UDP is a standard, low-overhead, connectionless, host-to-host protocol that is used over the IP.
- UDP allows an application program on one computer to send a datagram to an application program on another computer.

Introduction

- Java supports both stream-based and packet-based communications.
- Stream-based communications use TCP for data transmission, whereas packet-based communications use UDP.
- TCP can detect lost transmission and resubmit them, transmission are lossless and reliable. UDP cannot guarantee lossless transmission.
- Stream-based communications are used in most areas of Java Programming which we will focus in this module.

Client/Server Computing

- Networking is tightly integrated in Java where the Java API provides the classes for creating sockets to facilitate program communications over the Internet.
- Sockets are the endpoints of logical connections between two hosts and can be used to send and receive data.
- Java treats socket communications much as it treats I/O operations; thus, program can read from or write to sockets as easily as they can read from or write to files.

Client/Server Computing (cont.)

- Network programming usually involves a server and one or more clients.
  - The client sends requests to the server.
  - The server responds.

- The following is usually the process:
  - The client begins by attempting to establish a connection to the server.
  - The server can accept or deny the connection.
  - Once a connection is established, the client and the server communicate through sockets.

Server Sockets

- To establish a server, you need to create a server socket and attach it to a port, which is where the server listens for connections.

- The port identifies the TCP service on the socket.
  - Range 0 to 65536.
  - But port numbers 0 to 1024 are reserved for privileged services.
  - Example: email server runs on port 25 and Web server usually runs on port 80.

- You can choose any port number that is not currently used by other program.

Server Sockets (cont.)

- The following statement creates a server socket serverSocket:

```
ServerSocket serverSocket = new ServerSocket(port);
```

- Note: Attempting to create a server socket on a port already in use would cause a java.net.BindException

- After a server socket is created, the server can use the following statement to listen for connections (This statement waits until a client connects to the server socket):

```
Socket socket = serverSocket.accept();
```

Client Sockets

- The client issues the following statement to request a connection to a server (This statement opens a socket so that the client program can communicate with the server):

Socket socket = new Socket(serverName, port);

- Example:

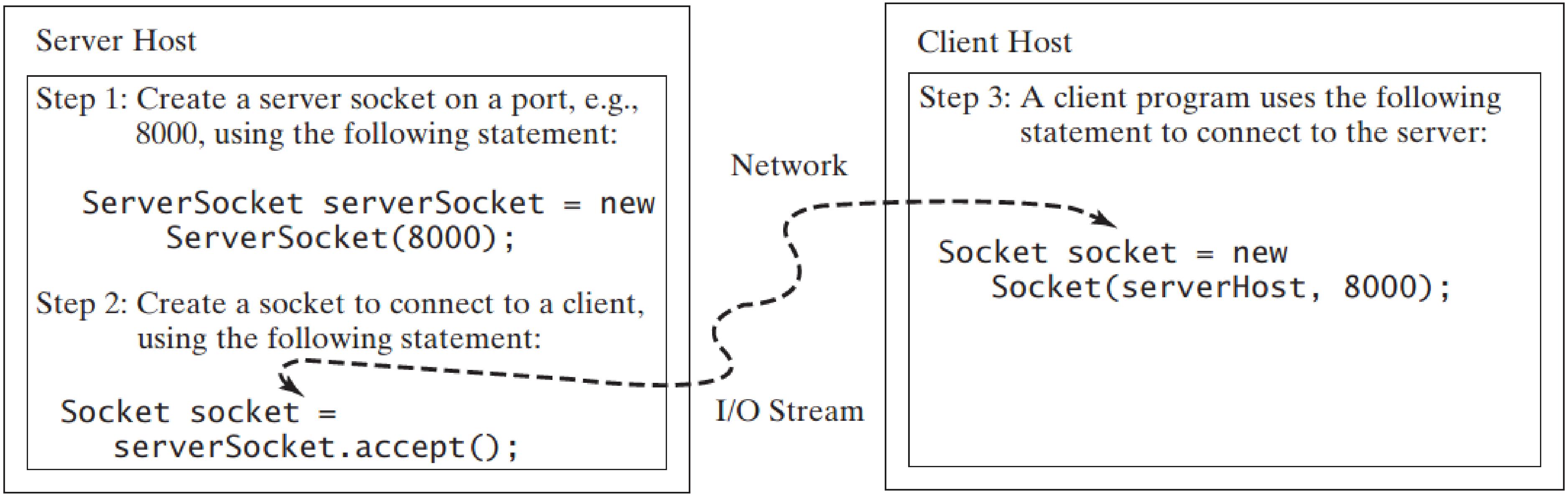
Socket socket = new Socket(“111.222.333.444”, 8000);

Socket socket = new Socket(“jailaniwebsite.com”, 8000);

- Note: Socket constructor throws a java.net.UnknownHostException if the host cannot be found.

Connection between Server & Client Sockets

- The following shows the process of the server creates a server socket and connects to the client with a client socket.



Server Host

Step 1: Create a server socket on a port, e.g., 8000, using the following statement:

```
ServerSocket serverSocket = new
    ServerSocket(8000);
```

Step 2: Create a socket to connect to a client, using the following statement:

```
Socket socket =
  serverSocket.accept();
```

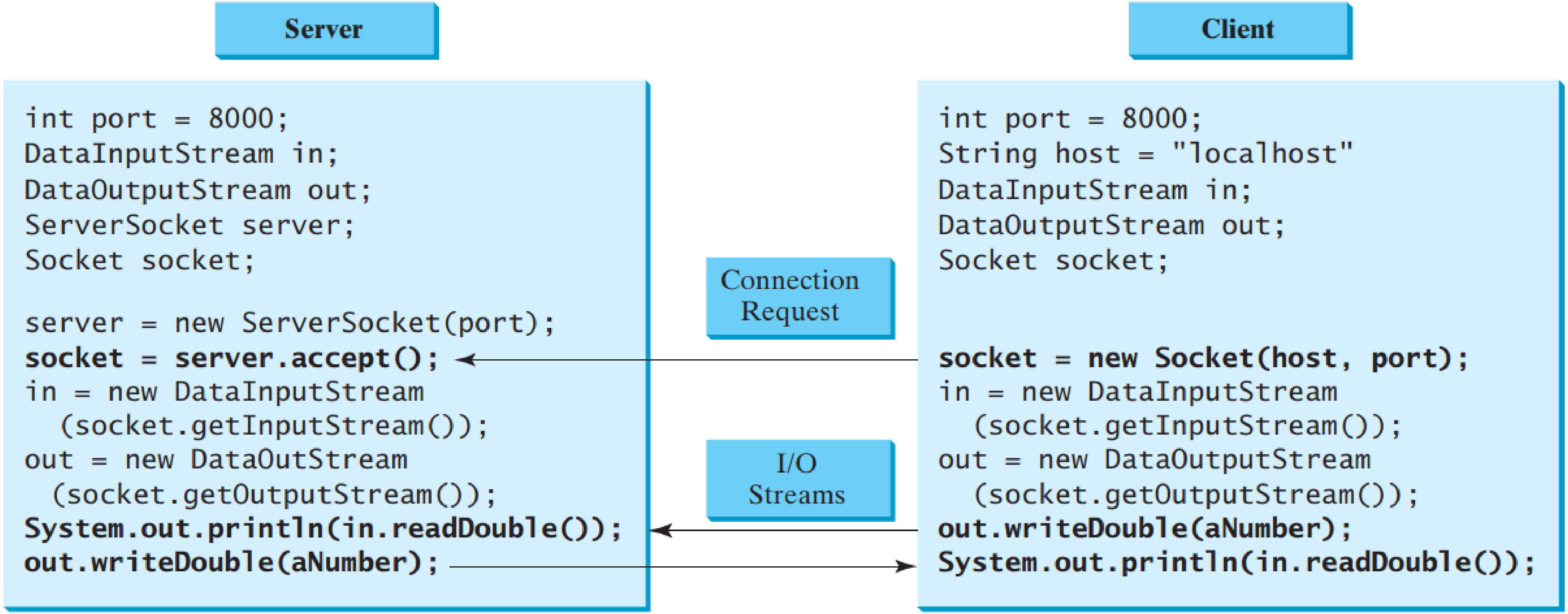
Client Host

Step 3: A client program uses the following statement to connect to the server:

```
Socket socket = new
  Socket(serverHost, 8000);
```

Data Transmission through Sockets

- After the server accepts the connection, communication between the server and the client is conducted in the same way as for I/O streams.



Server

```
int port = 8000;
DataInputStream in;
DataOutputStream out;
ServerSocket server;
Socket socket;

server = new ServerSocket(port);
socket = server.accept();
in = new DataInputStream
  (socket.getInputStream());
out = new DataOutStream
  (socket.getOutputStream());
System.out.println(in.readDouble());
out.writeDouble(aNumber);
```

Client

```
int port = 8000;
String host = "localhost"
DataInputStream in;
DataOutputStream out;
Socket socket;


socket = new Socket(host, port);
in = new DataInputStream
  (socket.getInputStream());
out = new DataOutputStream
  (socket.getOutputStream());
out.writeDouble(aNumber);
System.out.println(in.readDouble());
```

Data Transmission through Sockets (cont.)

- To get an input stream use the getInputStream() method on a socket object.

```
InputStream input = socket.getInputStream();
```

- To get an output stream use the getOutputStream() method on a socket object.

```
OutputStream output = socket.getOutputStream();
```

- The InputStream and OutputStream streams are used to read or write bytes.

Data Transmission through Sockets (cont.)

- You can use DataInputStream, DataOutputStream, BufferedReader, and PrintWriter to wrap on the InputStream and OutputStream to read or write data, such as int, double or String.

- Example statements to create stream to read and write primitive data values:

```
DataInputStream input = new
DataInputStream(socket.getInputStream());

DataOutputStream output = new
DataOutputStream(socket.getOutputStream());
```

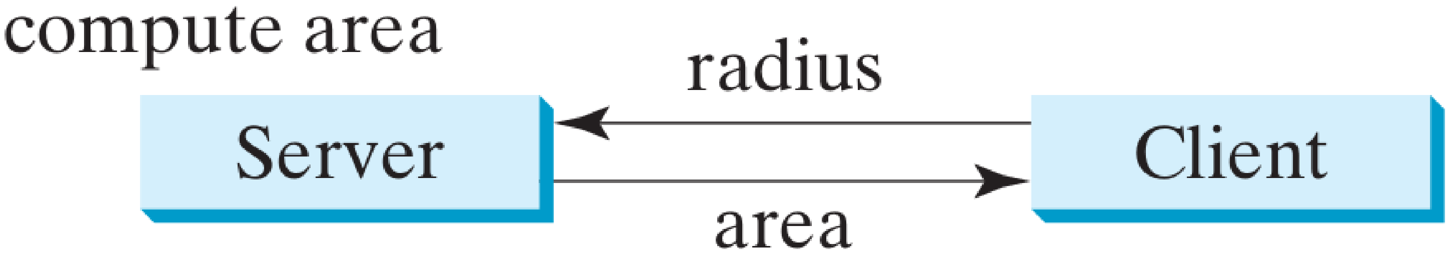
Data Transmission through Sockets (cont.)

- The server can use:
  - **input.readDouble()** to receive a **double** value from the client.
  - **output.writeDouble(d)** to send the **double** value **d** to the client.

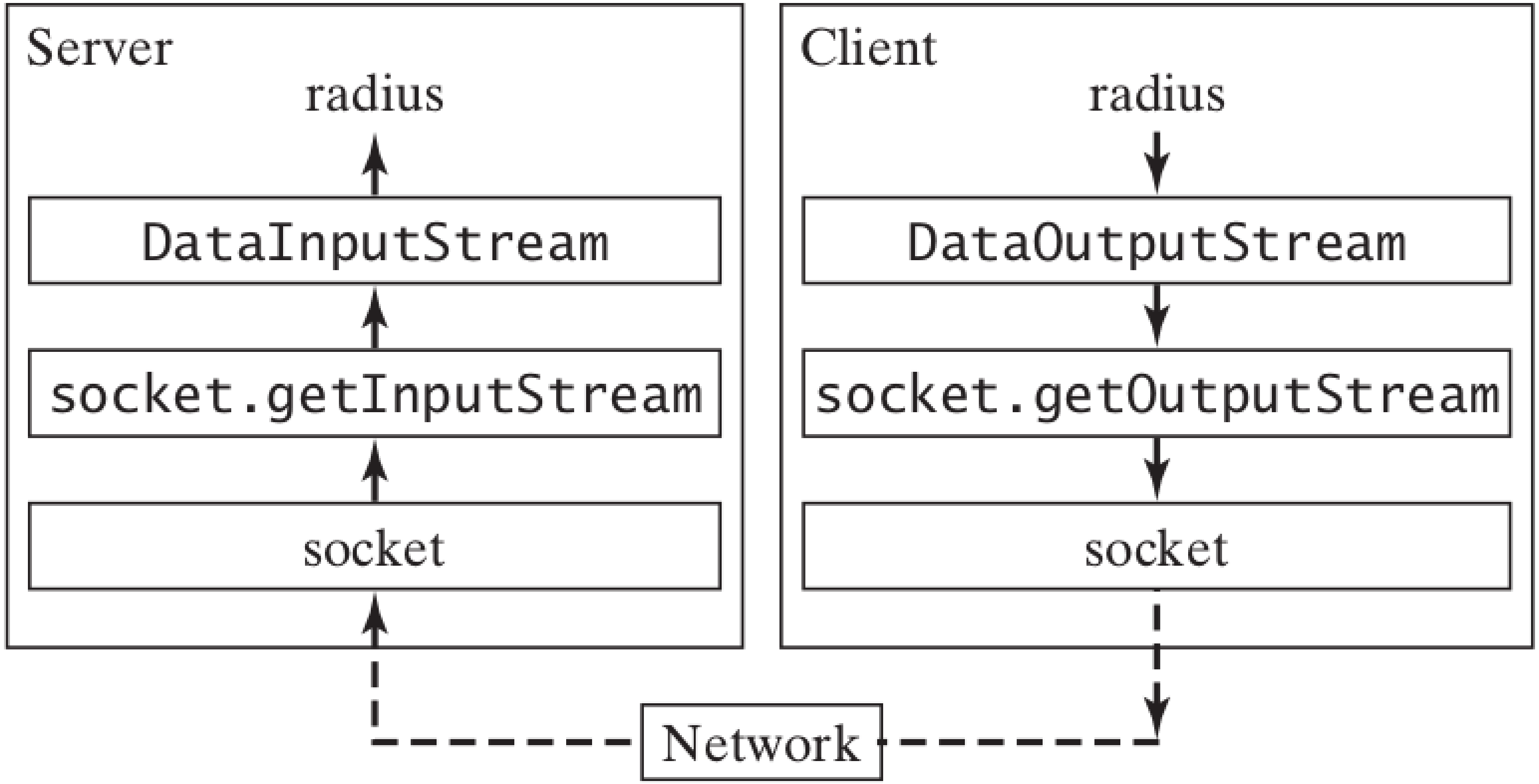
- Tip: Recall that binary I/O is more efficient than text I/O because text I/O requires encoding and decoding. Therefore, it is better to use binary I/O for transmitting data between a server and a client to improve performance.

Example: A Client/Server

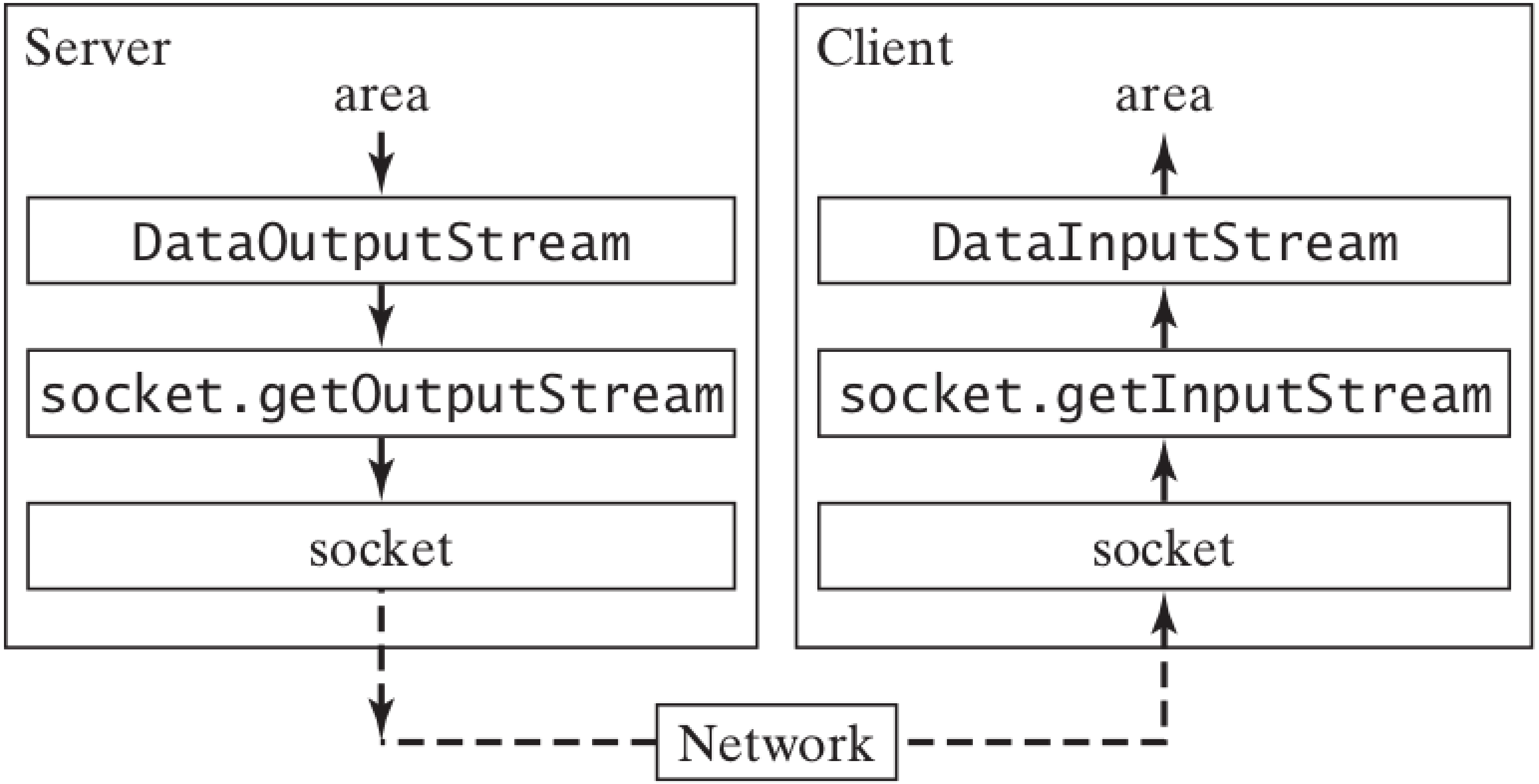
- This example presents a client program and a server program.
  - The client sends data (radius of a circle) to a server.
  - The server receives the data (radius of a circle), uses it to produce a result (area of the circle) and then sends the result back to the client.
  - The client displays the result (area of the circle) on the console.



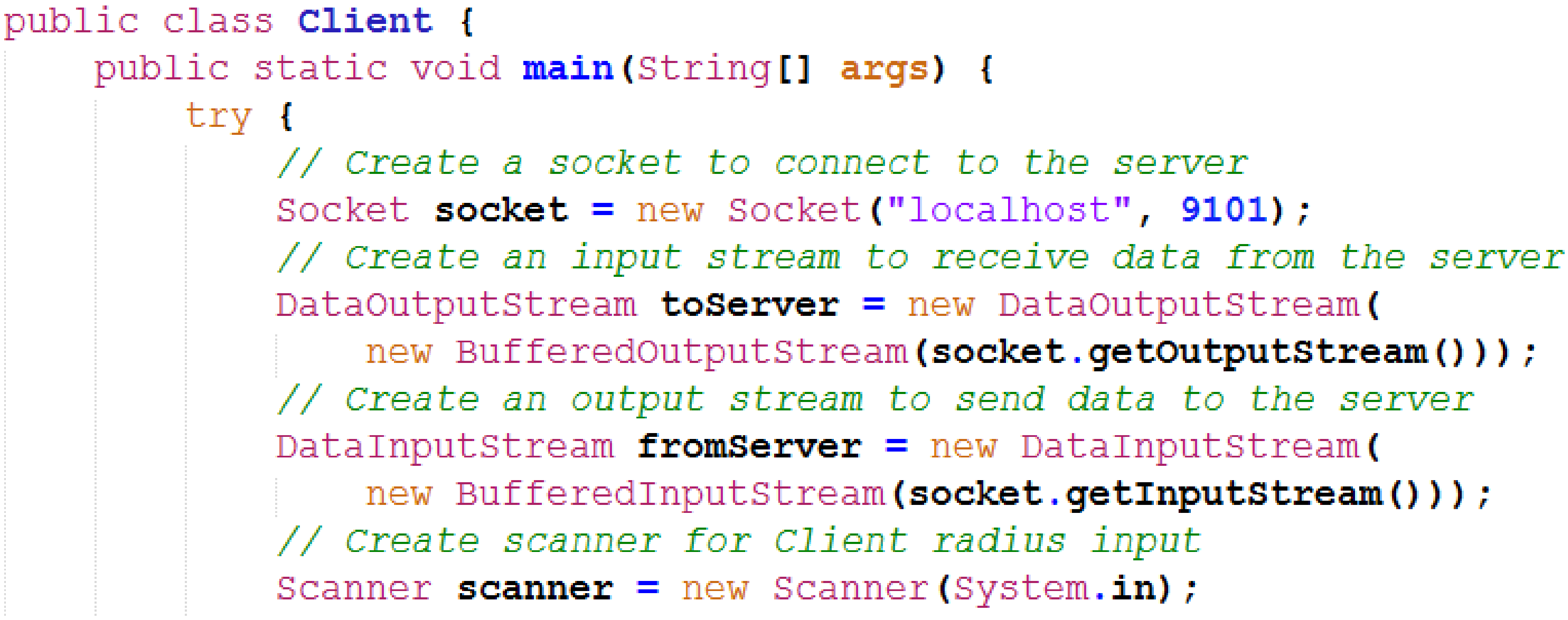
Example: A Client/Server (cont.)



Example: A Client/Server (cont.)

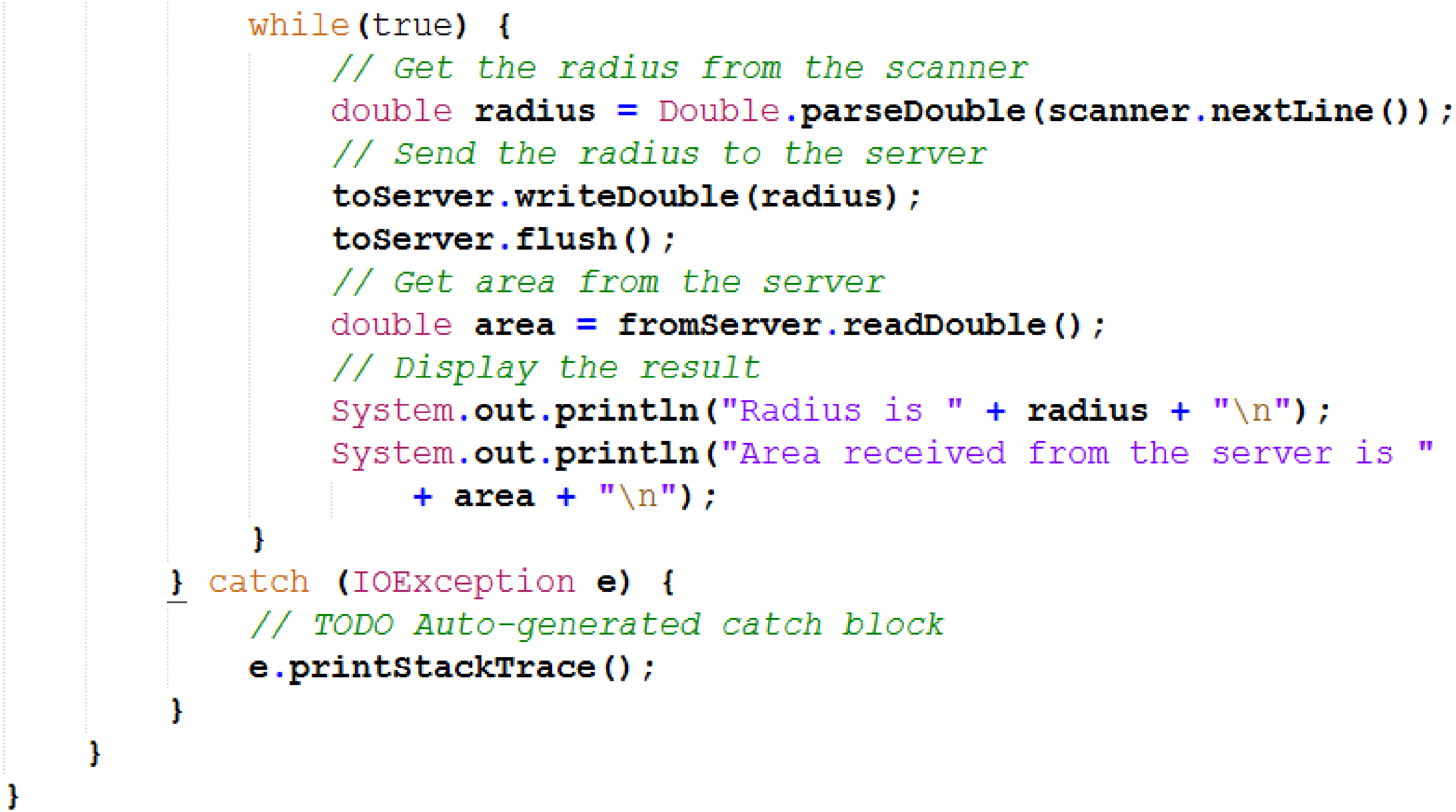


Example: Client.java



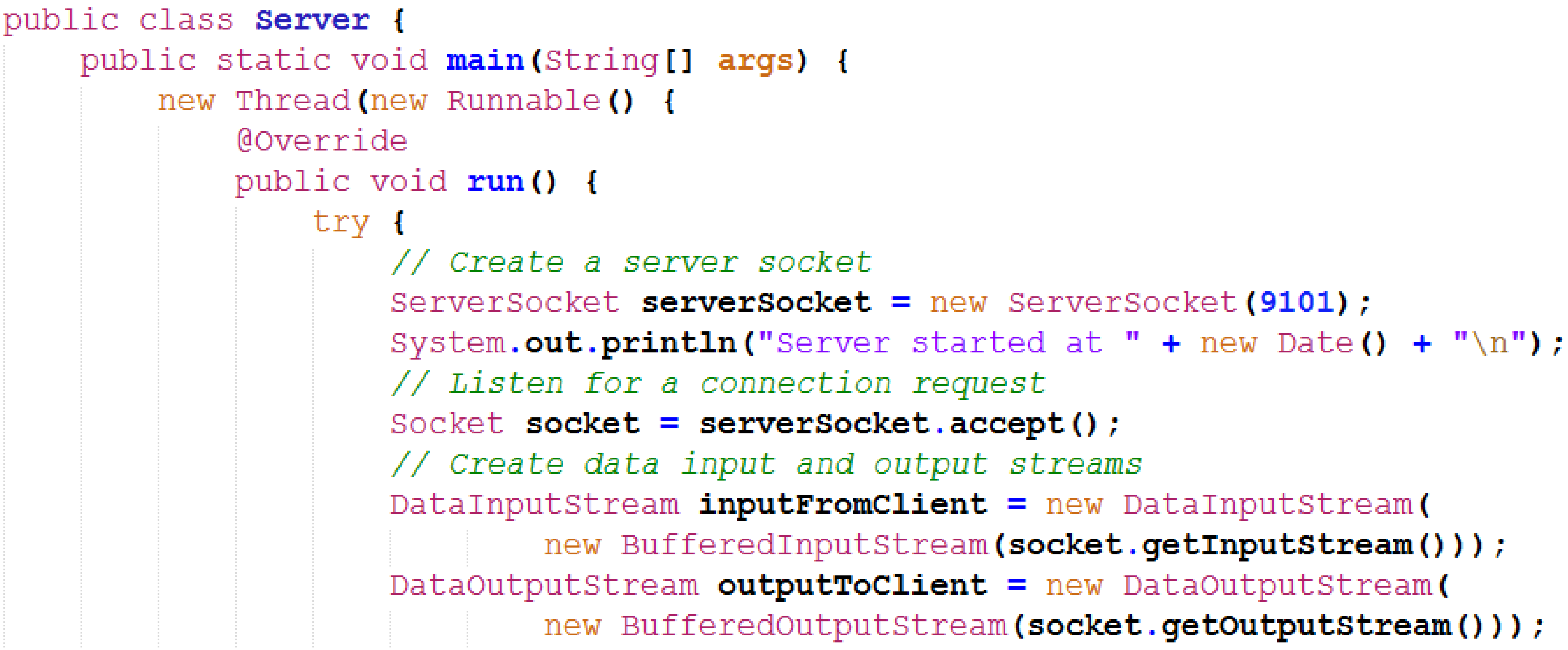
```
public class Client {
    public static void main(String[] args) {
        try {
            // Create a socket to connect to the server
            Socket socket = new Socket("localhost", 9101);
            // Create an input stream to receive data from the server
            DataOutputStream toServer = new DataOutputStream(
                new BufferedOutputStream(socket.getOutputStream()));
            // Create an output stream to send data to the server
            DataInputStream fromServer = new DataInputStream(
                new BufferedInputStream(socket.getInputStream()));
            // Create scanner for Client radius input
            Scanner scanner = new Scanner(System.in);
```

Example: Client.java (cont.)



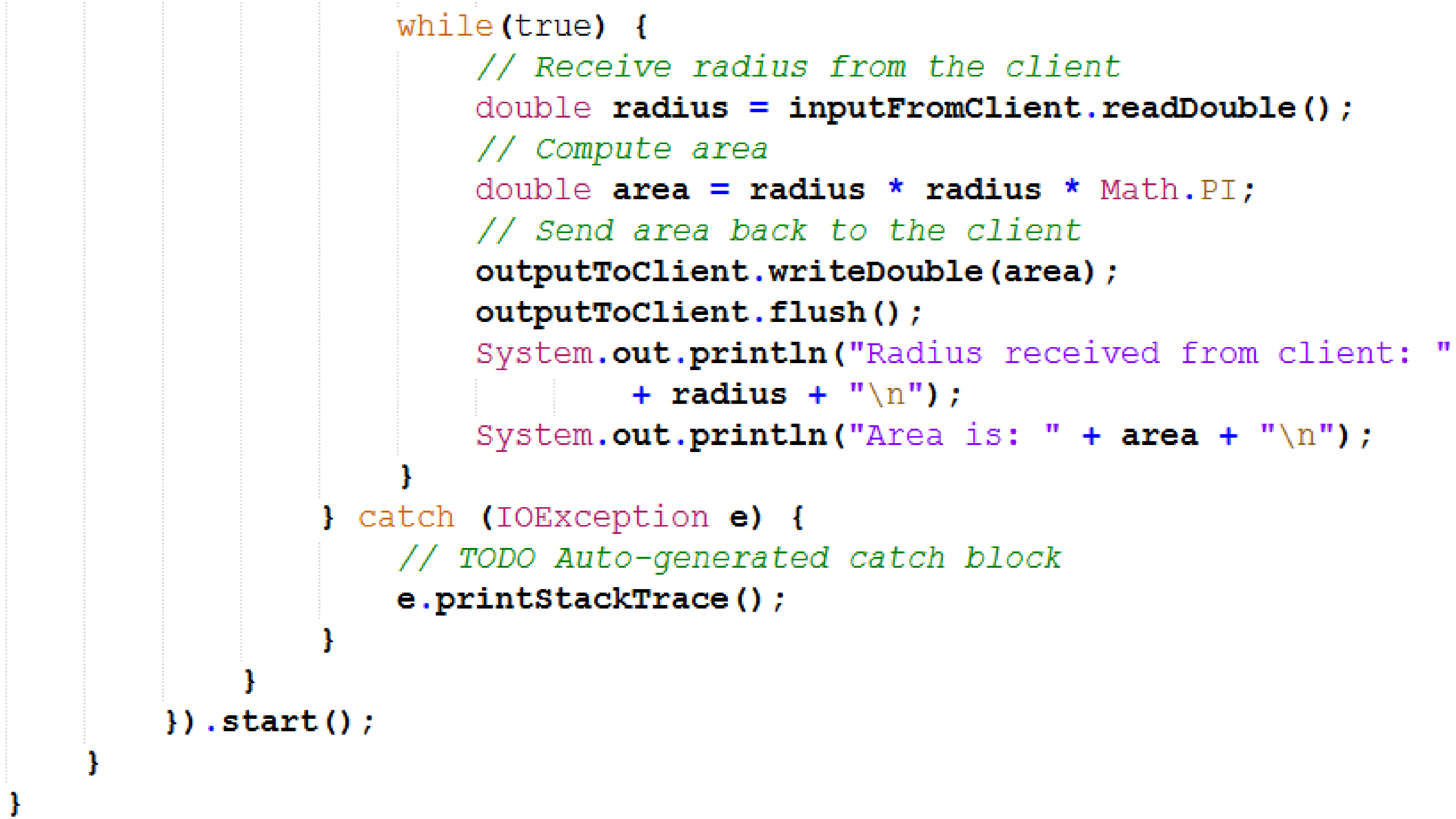
```java
            while(true) {
                // Get the radius from the scanner
                double radius = Double.parseDouble(scanner.nextLine());
                // Send the radius to the server
                toServer.writeDouble(radius);
                toServer.flush();
                // Get area from the server
                double area = fromServer.readDouble();
                // Display the result
                System.out.println("Radius is " + radius + "\n");
                System.out.println("Area received from the server is "
                    + area + "\n");
            }
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

Example: Server.java



```java
public class Server {
    public static void main(String[] args) {
        new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    // Create a server socket
                    ServerSocket serverSocket = new ServerSocket(9101);
                    System.out.println("Server started at " + new Date() + "\n");
                    // Listen for a connection request
                    Socket socket = serverSocket.accept();
                    // Create data input and output streams
                    DataInputStream inputFromClient = new DataInputStream(
                            new BufferedInputStream(socket.getInputStream()));
                    DataOutputStream outputToClient = new DataOutputStream(
                            new BufferedOutputStream(socket.getOutputStream()));
```

Example: Server.java (cont.)



```
                    while(true) {
                        // Receive radius from the client
                        double radius = inputFromClient.readDouble();
                        // Compute area
                        double area = radius * radius * Math.PI;
                        // Send area back to the client
                        outputToClient.writeDouble(area);
                        outputToClient.flush();
                        System.out.println("Radius received from client: "
                                + radius + "\n");
                        System.out.println("Area is: " + area + "\n");
                    }
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        }).start();
    }
}
```

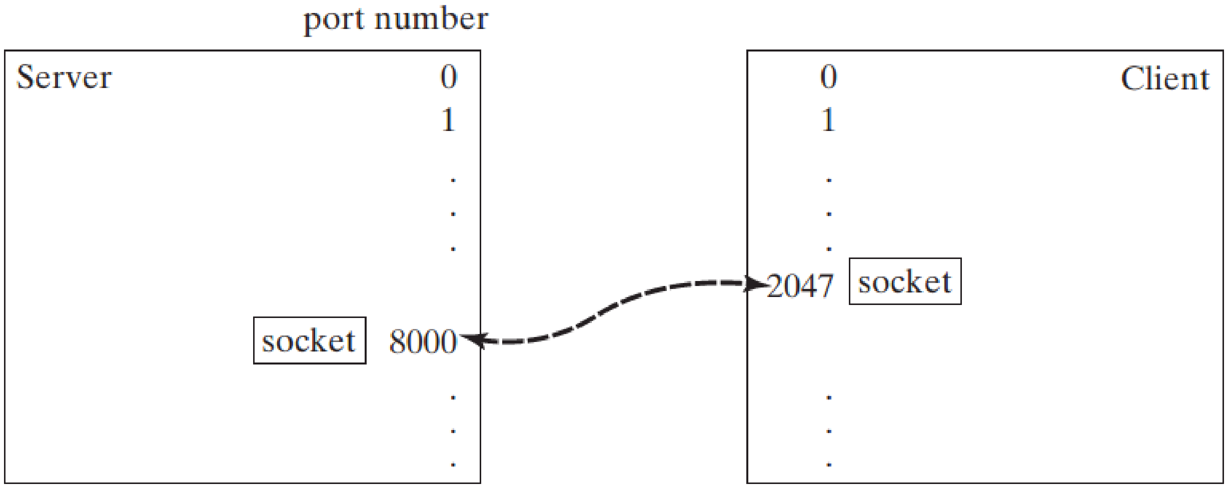
Example: A Client/Server (cont.)

- If the server is not running, the client program terminates with a ConnectException.

- If you receive a BindException when you start the server, the server port is currently in use.

Example: A Client/Server (cont.)

- Note:
  - When you create a server socket, you have to specify a port (e.g. 8000) for the socket.
  - When a client connects to the server, a socket is created on the client. This socket has its own local port.
  - This port (e.g. 2047) is automatically chosen by the JVM.



InetAddress

- Occasionally, you would like to know who is connecting to the server.

- You can use the InetAddress class to find the client’s host name and IP address.

- The InetAddress class models an IP address.

InetAddress (cont.)

- You can use the following statement in the server program to get an instance of InetAddress on a socket that connects to the client.

```
InetAddress inetAddress = socket.getInetAddress();
```

- Next, you can display the client’s host name and IP address, as follows.

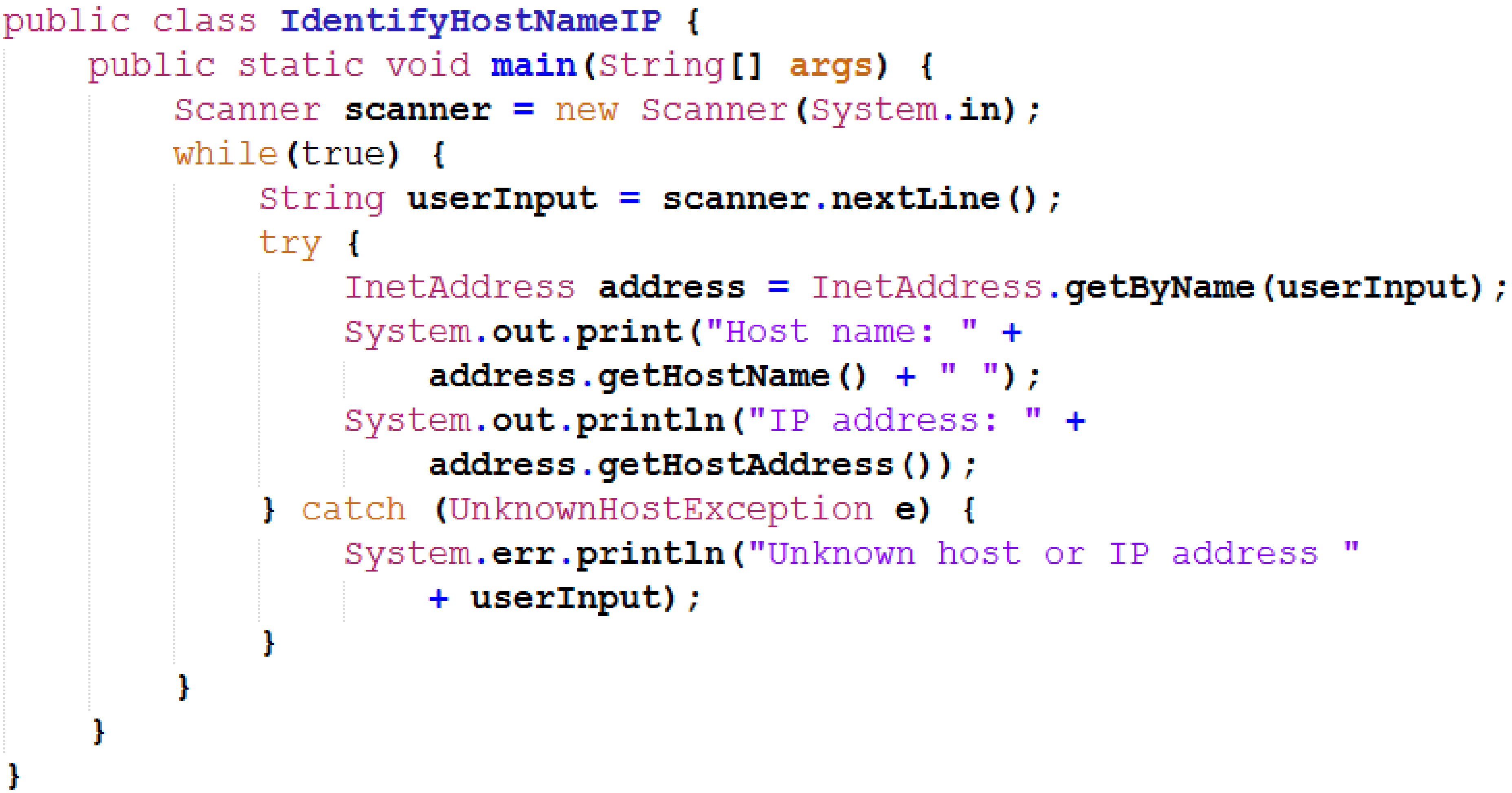
```
System.out.println(“Client’s host name is ” +
inetAddress.getHostName());
System.out.println(“Client’s IP address is ” +
inetAddress.getHostAddress());
```

InetAddress (cont.)

- You can also create an instance of InetAddress from a host name or IP address using static getByName method.

```
InetAddress address =
InetAddress.getByName(“jailanirahman.com”);
```

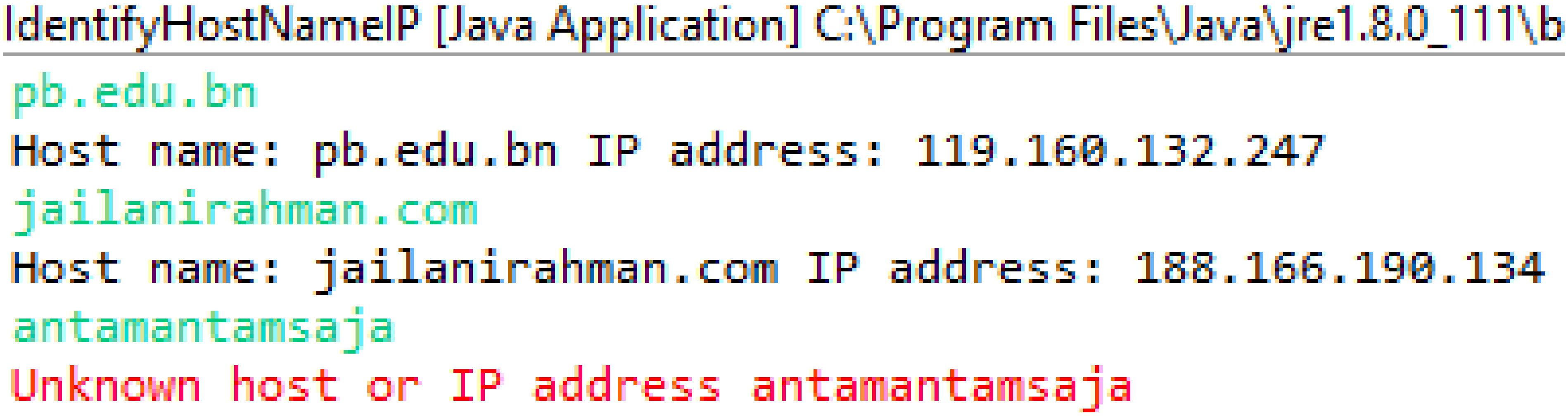
Example: IdentifyHostNameIP.java



```java
public class IdentifyHostNameIP {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        while(true) {
            String userInput = scanner.nextLine();
            try {
                InetAddress address = InetAddress.getByName(userInput);
                System.out.print("Host name: " +
                    address.getHostName() + " ");
                System.out.println("IP address: " +
                    address.getHostAddress());
            } catch (UnknownHostException e) {
                System.err.println("Unknown host or IP address "
                    + userInput);
            }
        }
    }
}
```

Example: IdentifyHostNameIP.java (cont.)

- The application will asked for input due to the Scanner and it will output the host name and ip address.



```
IdentifyHostNameIP [Java Application] C:\Program Files\Java\jre1.8.0_111\b
pb.edu.bn
Host name: pb.edu.bn IP address: 119.160.132.247
jailanirahman.com
Host name: jailanirahman.com IP address: 188.166.190.134
antamantamsaja
Unknown host or IP address antamantamsaja
```

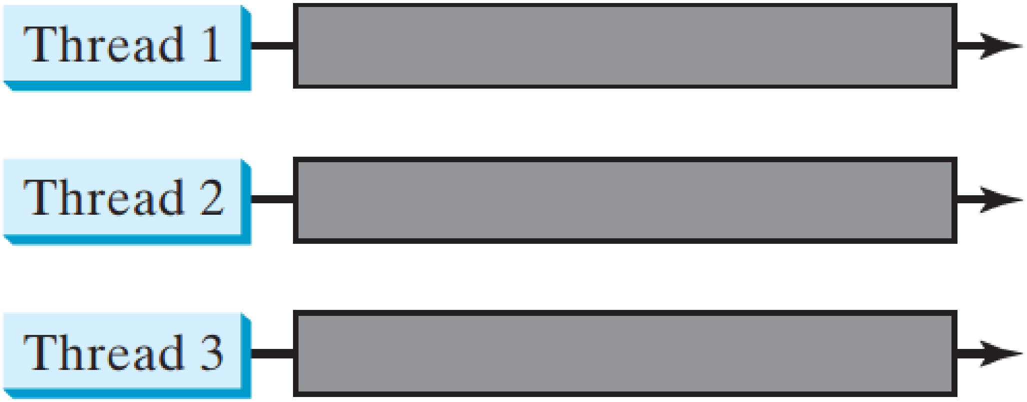
Serving Multiple Clients

- Multiple clients are quite often connected to a single server at the same time.
- Typically, a server runs continuously on a server computer, and clients from all over the Internet can connect to it.
- You can use threads to handle the server’s multiple clients simultaneously.

Threads

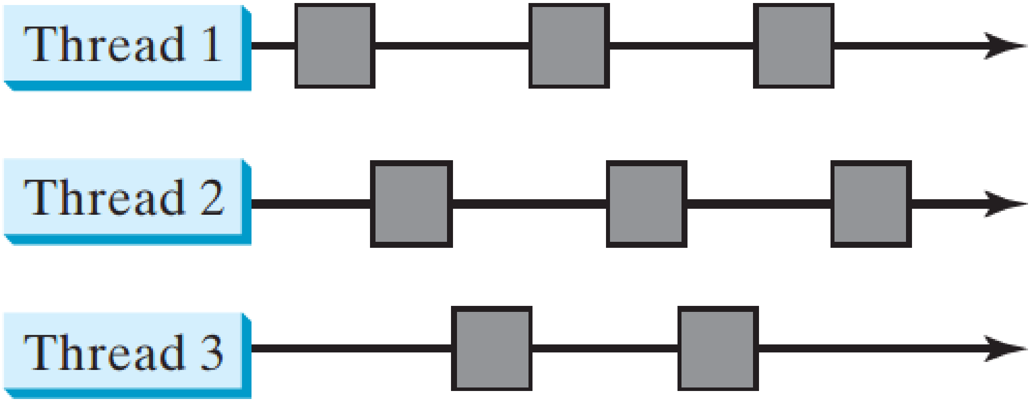
- A thread provides the mechanism for running a task. With Java, you can launch multiple threads from a program concurrently.

- The following is an example of multiple threads running on multiple CPUs:



Threads (cont.)

- The following is an example of multiple threads share a single CPU:



- In a single processor systems, the multiple threads share CPU time, known as time sharing and the operating system is responsible for scheduling and allocating resources to them.

Multiple threads

- Multithreading can make your program more responsive and interactive, as well as enhance performance.
- Example: A good word processor lets you print or save a file while you are typing.
- Java provides exceptionally good support for creating and running threads and for locking resources to prevent conflicts.
- You can create additional threads to run concurrent tasks in the program.

Java Task

- **Tasks are objects**. You can implement constructors when you are defining a task class.

- To create tasks, you have to first define a class for tasks, which implements the **Runnable Interface**.

```
public class TaskClass implements Runnable { …. }
```

- Then you need to implement **run method** to tell the system how your thread is going to run.

```
public void run() { …. }
```

Java Task (cont.)

- Once you have defines a TaskClass, you can create a task using its constructor.

```
TaskClass task = new TaskClass(…);
```

- Then the task created must be executed in a thread.

Java Thread

- The Thread class contains the constructor for creating threads and many useful methods for controlling threads.

- To create a thread for a task:

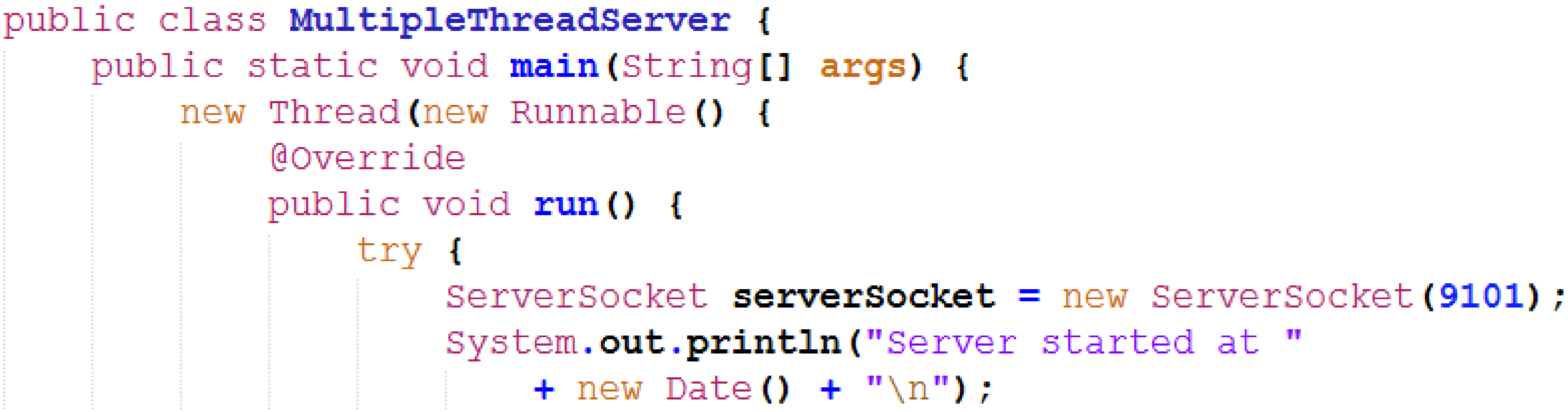
```
Thread thread = new Thread(task);
```

- You can then invoke the **start() method** to tell the JVM that the thread is ready to run and execute the task by invoking the **task's run() method**.

```
Thread.start();
```

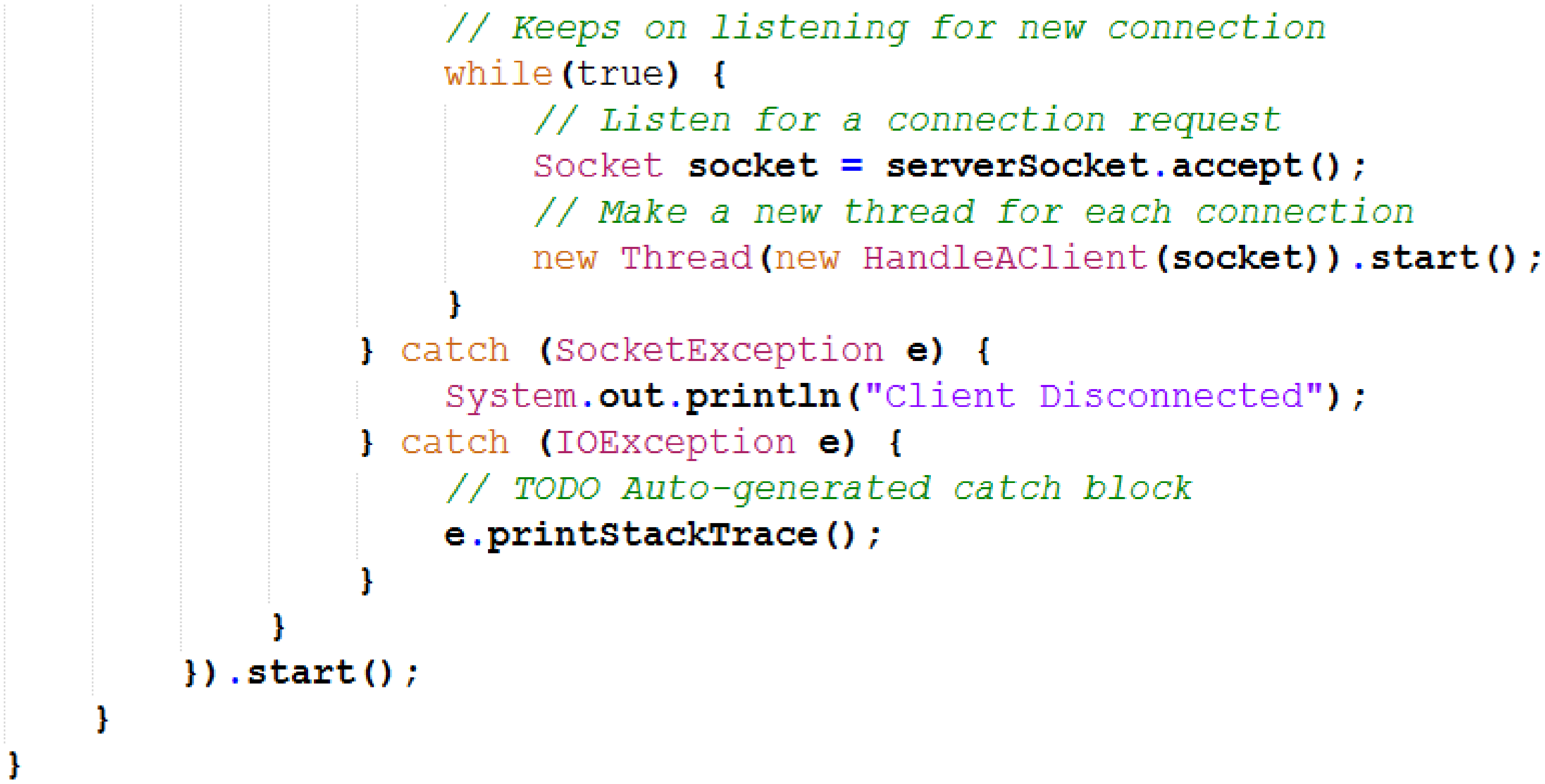
Example: MultipleThreadServer.java

- Lets implement another server that can handle multiple clients.



```java
public class MultipleThreadServer {
    public static void main(String[] args) {
        new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    ServerSocket serverSocket = new ServerSocket(9101);
                    System.out.println("Server started at "
                        + new Date() + "\n");
```

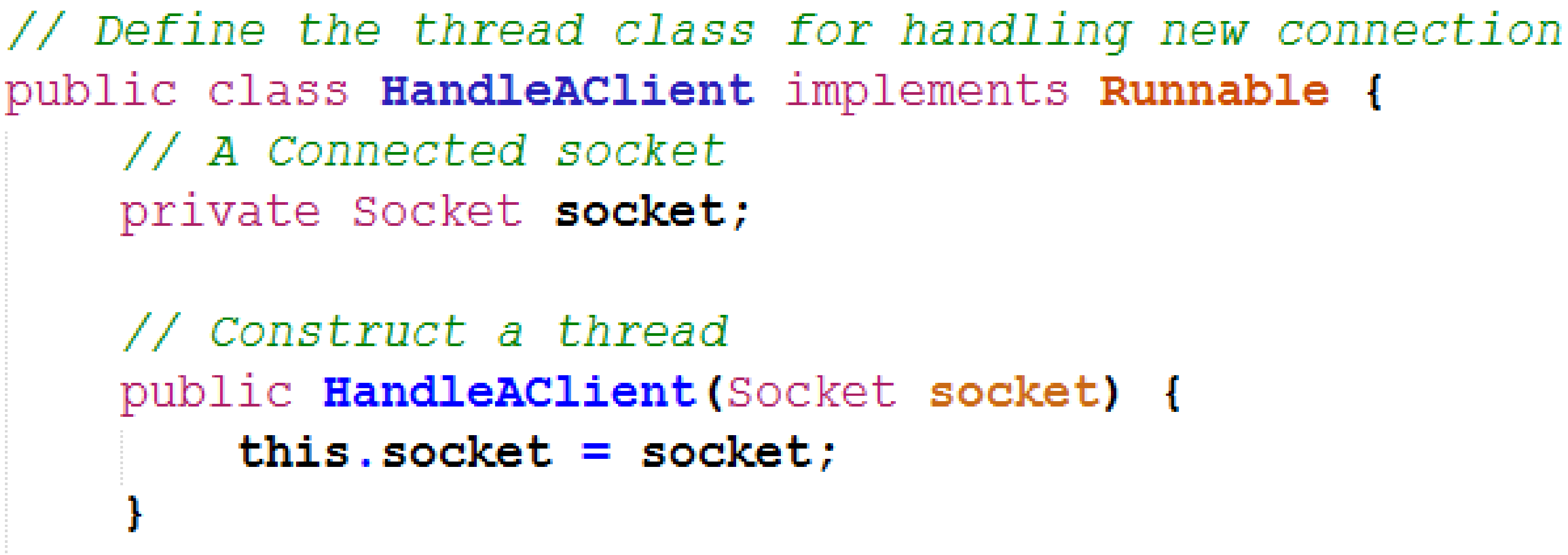
Example: MultipleThreadServer.java (cont.)



```
                    // Keeps on listening for new connection
                    while(true) {
                        // Listen for a connection request
                        Socket socket = serverSocket.accept();
                        // Make a new thread for each connection
                        new Thread(new HandleAClient(socket)).start();
                    }
                } catch (SocketException e) {
                    System.out.println("Client Disconnected");
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        }).start();
    }
}
```

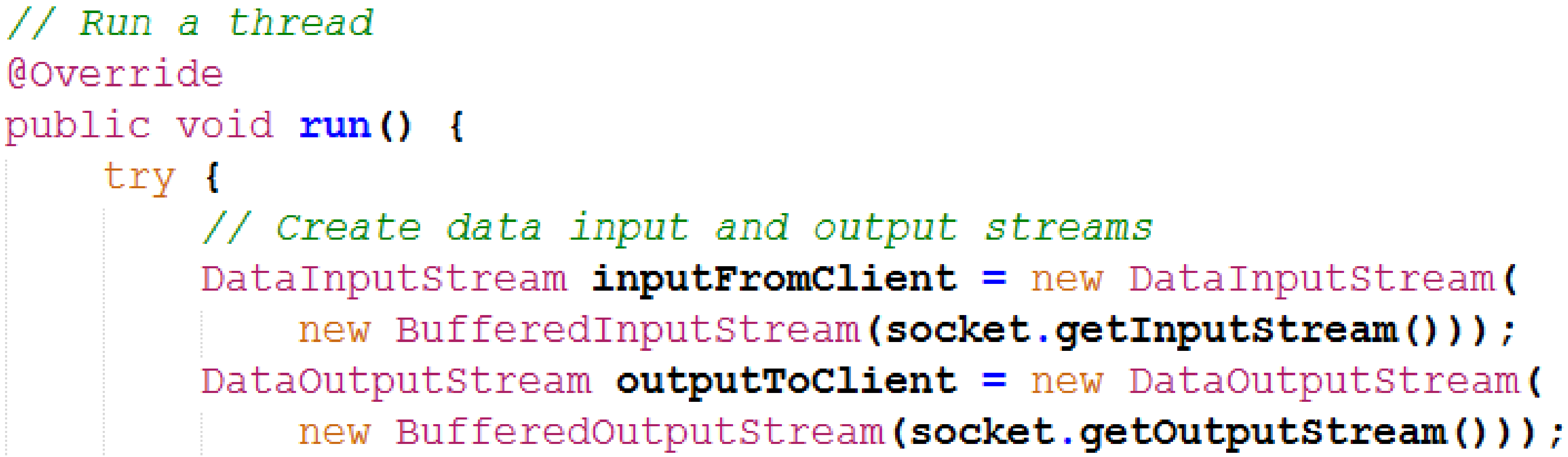
Example: HandleAClient.java

- This class will handle each client connected to the server.



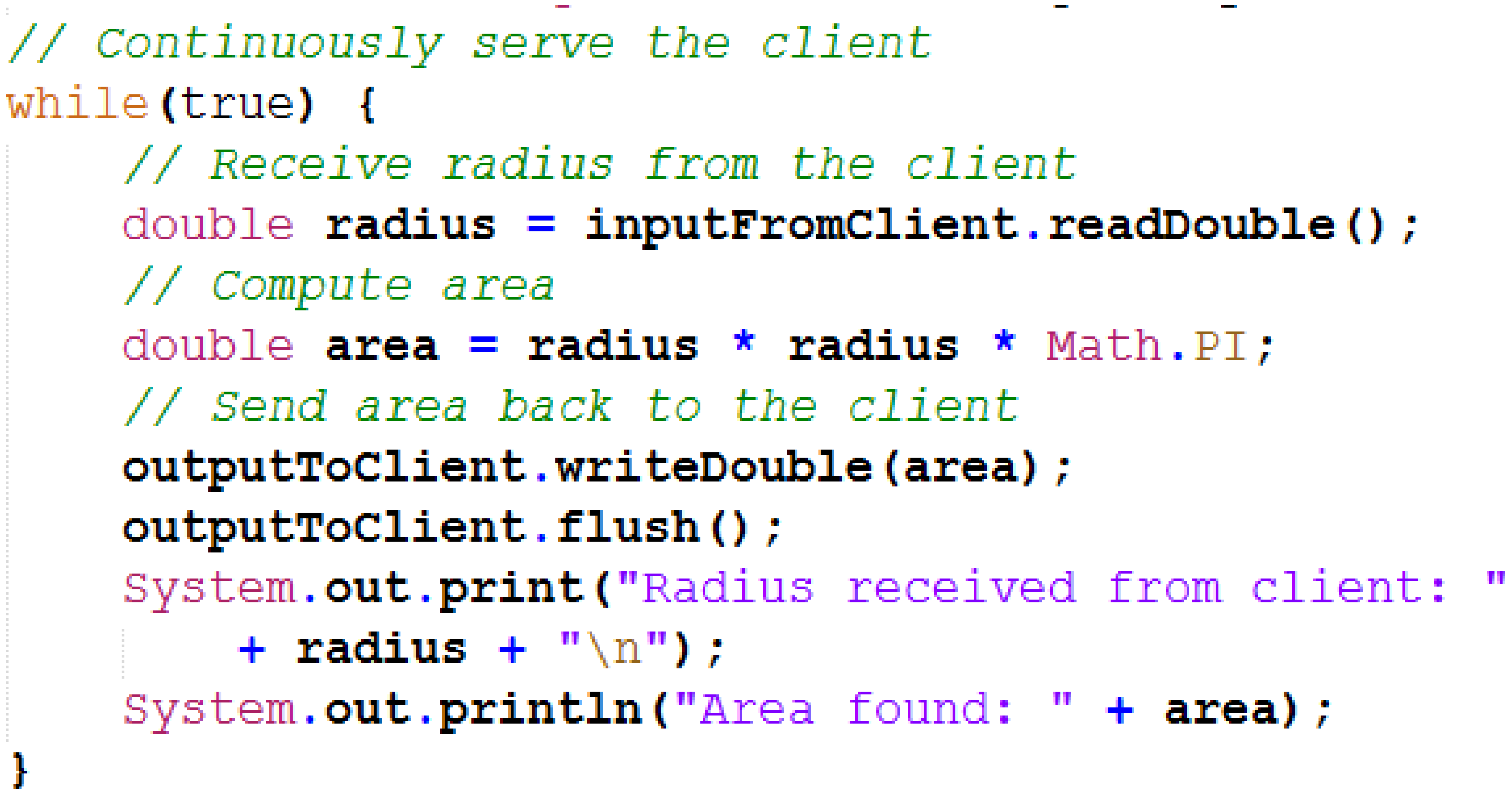
```
// Define the thread class for handling new connection
public class HandleAClient implements Runnable {
    // A Connected socket
    private Socket socket;

    // Construct a thread
    public HandleAClient(Socket socket) {
        this.socket = socket;
    }
```

Example: HandleAClient.java (cont.)



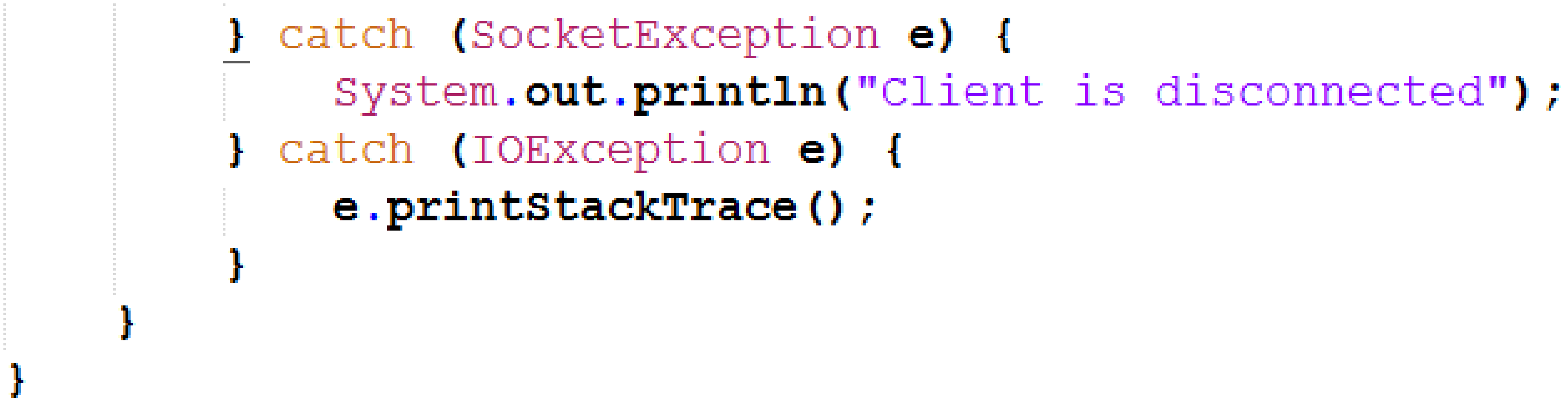
```
// Run a thread
@Override
public void run() {
    try {
        // Create data input and output streams
        DataInputStream inputFromClient = new DataInputStream(
            new BufferedInputStream(socket.getInputStream()));
        DataOutputStream outputToClient = new DataOutputStream(
            new BufferedOutputStream(socket.getOutputStream()));
```

Example: HandleAClient.java (cont.)



```
// Continuously serve the client
while(true) {
    // Receive radius from the client
    double radius = inputFromClient.readDouble();
    // Compute area
    double area = radius * radius * Math.PI;
    // Send area back to the client
    outputToClient.writeDouble(area);
    outputToClient.flush();
    System.out.print("Radius received from client: "
        + radius + "\n");
    System.out.println("Area found: " + area);
}
```

Example: HandleAClient.java (cont.)



```
        } catch (SocketException e) {
            System.out.println("Client is disconnected");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

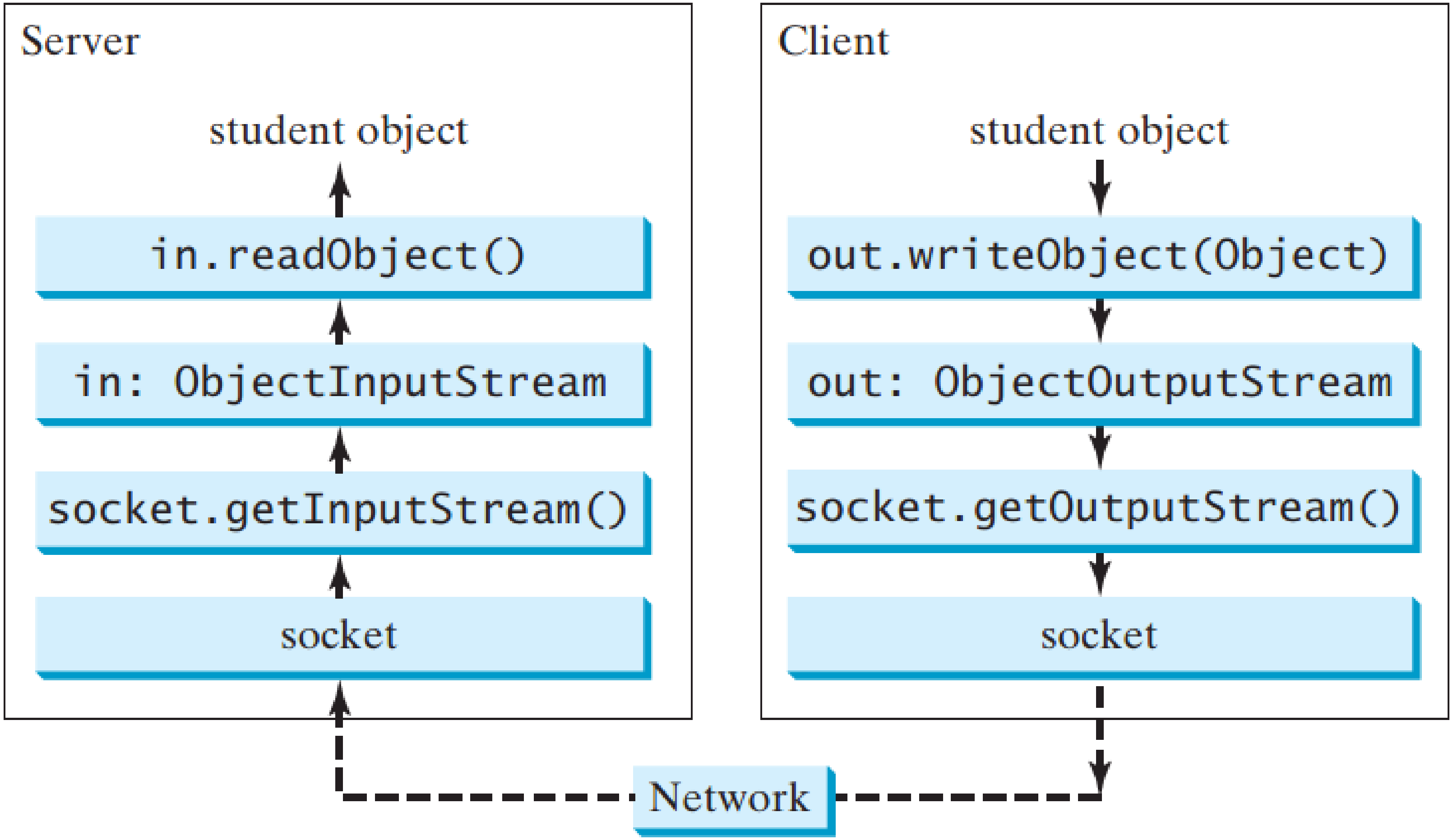
Sending and Receiving Object

- In the preceding examples, you learned how to send and receive data of primitive types.

- You can also send and receive objects using ObjectOutputStream and ObjectInputStream on socket streams.

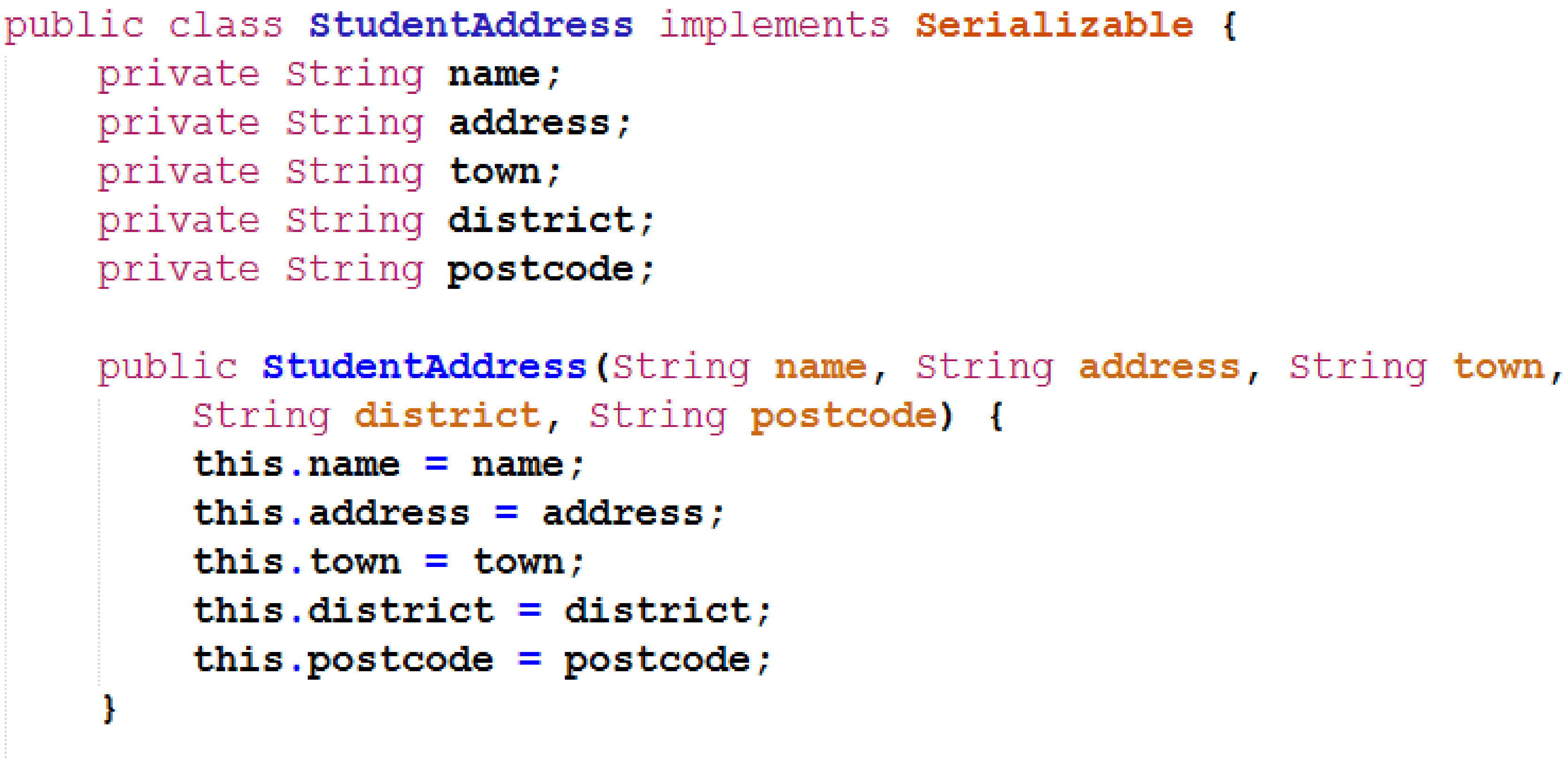
- To enable passing, the objects must be serializable.

Sending and Receiving Object (cont.)



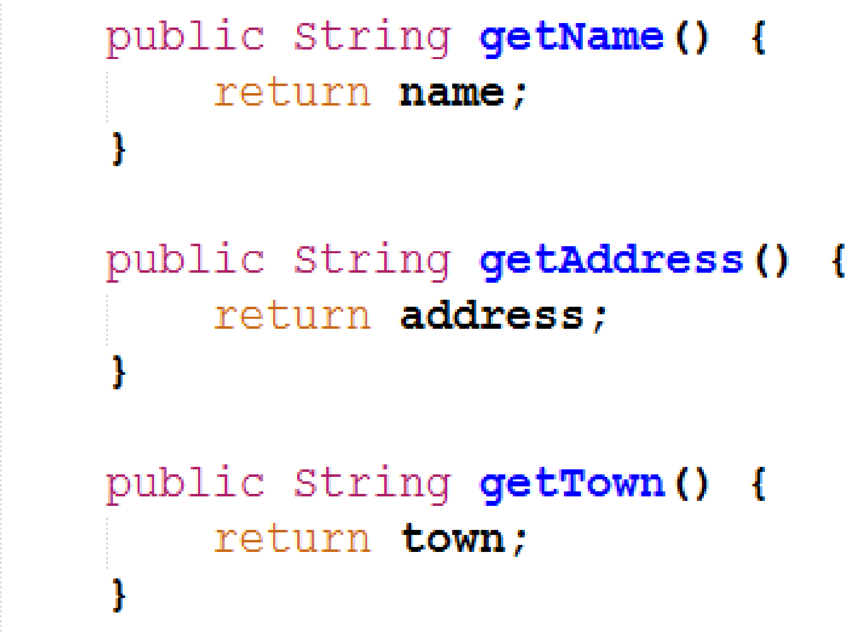
Example: StudentAddress.java

- This will be the object that going to be sent and received through the stream.

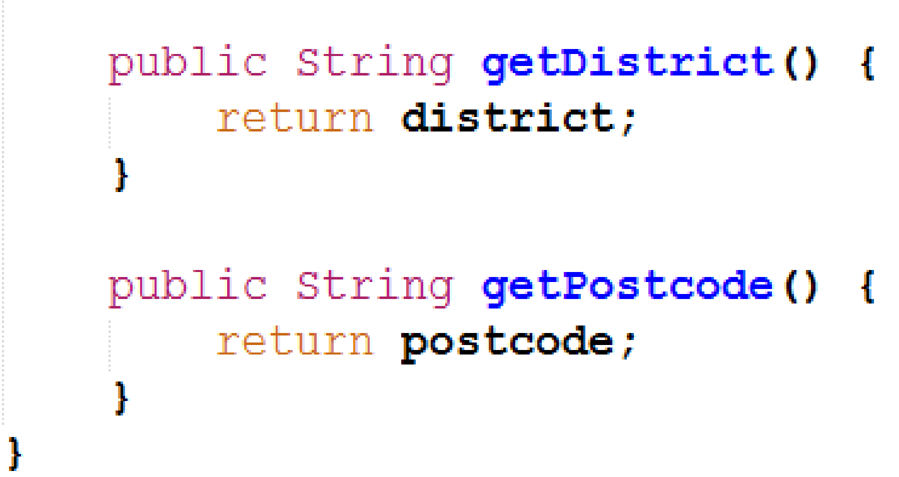


```java
public class StudentAddress implements Serializable {
    private String name;
    private String address;
    private String town;
    private String district;
    private String postcode;

    public StudentAddress(String name, String address, String town,
        String district, String postcode) {
        this.name = name;
        this.address = address;
        this.town = town;
        this.district = district;
        this.postcode = postcode;
    }
```

Example: StudentAddress.java (cont.)



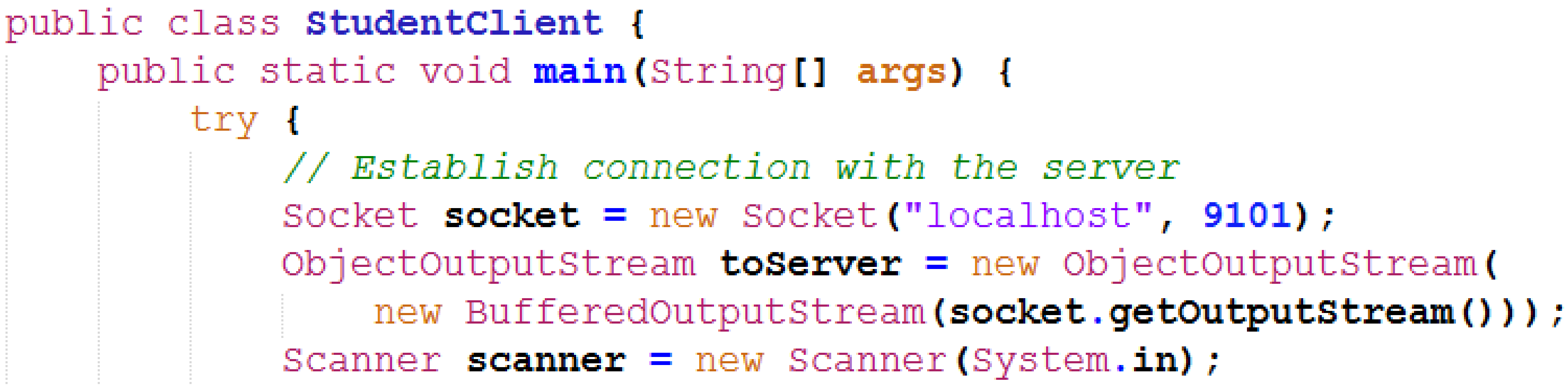
```
    public String getName() {
        return name;
    }

    public String getAddress() {
        return address;
    }

    public String getTown() {
        return town;
    }
```



```
    public String getDistrict() {
        return district;
    }

    public String getPostcode() {
        return postcode;
    }
}
```

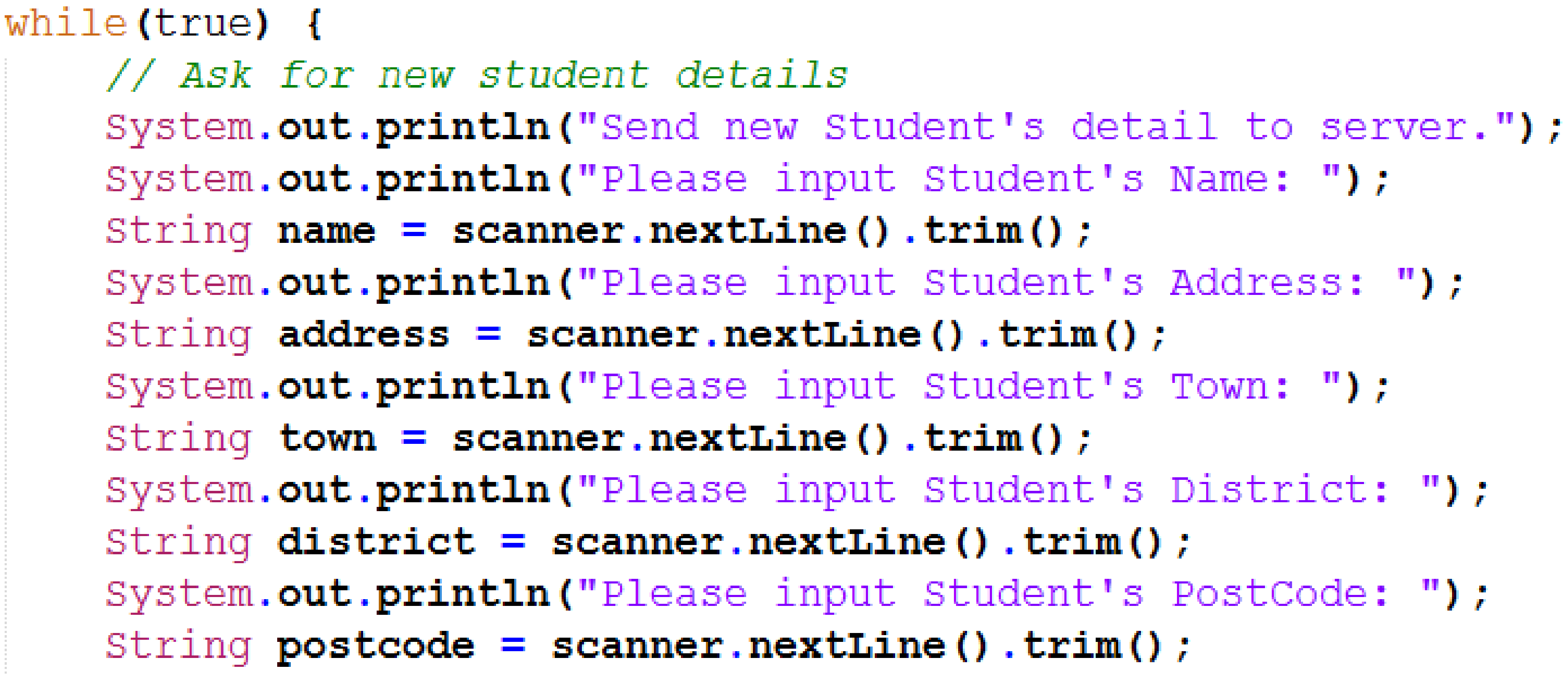
Example: StudentClient.java

- This client application will asked for student details and will send object with the student details to the server.



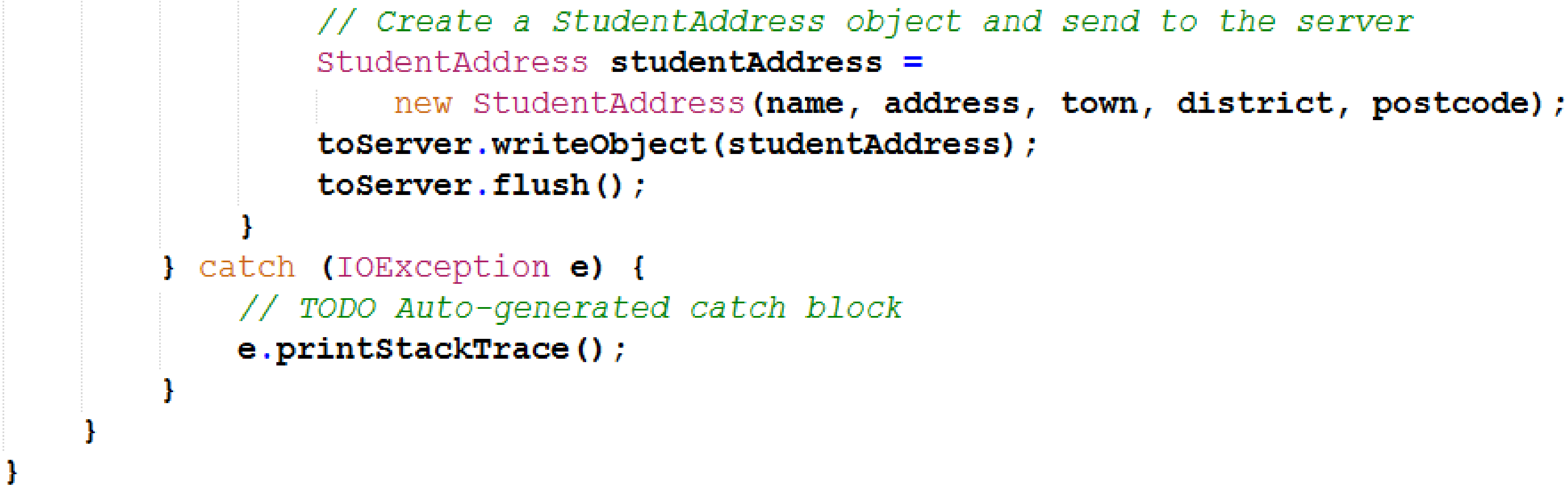
```
public class StudentClient {
    public static void main(String[] args) {
        try {
            // Establish connection with the server
            Socket socket = new Socket("localhost", 9101);
            ObjectOutputStream toServer = new ObjectOutputStream(
                new BufferedOutputStream(socket.getOutputStream()));
            Scanner scanner = new Scanner(System.in);
```

Example: StudentClient.java (cont.)



```java
while(true) {
    // Ask for new student details
    System.out.println("Send new Student's detail to server.");
    System.out.println("Please input Student's Name: ");
    String name = scanner.nextLine().trim();
    System.out.println("Please input Student's Address: ");
    String address = scanner.nextLine().trim();
    System.out.println("Please input Student's Town: ");
    String town = scanner.nextLine().trim();
    System.out.println("Please input Student's District: ");
    String district = scanner.nextLine().trim();
    System.out.println("Please input Student's PostCode: ");
    String postcode = scanner.nextLine().trim();
```

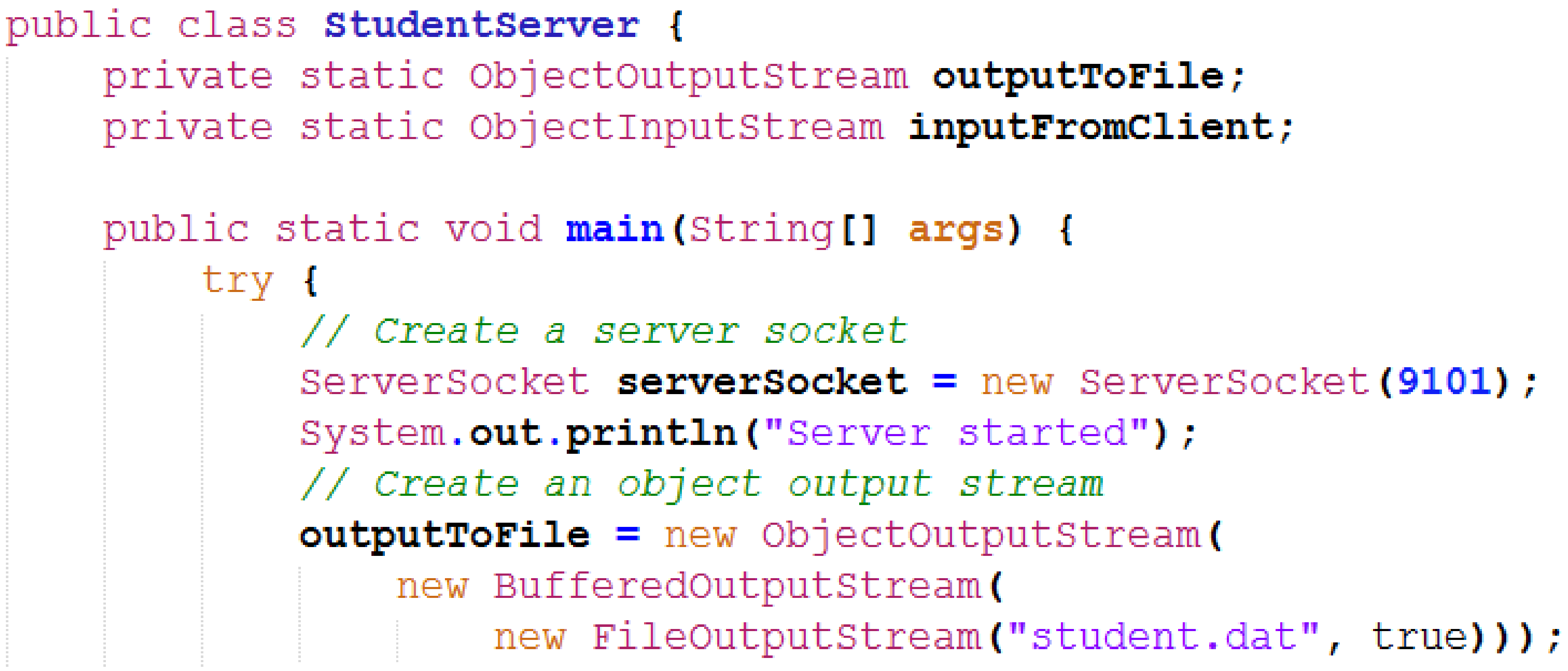
Example: StudentClient.java (cont.)



```
                // Create a StudentAddress object and send to the server
                StudentAddress studentAddress =
                    new StudentAddress(name, address, town, district, postcode);
                toServer.writeObject(studentAddress);
                toServer.flush();
            }
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

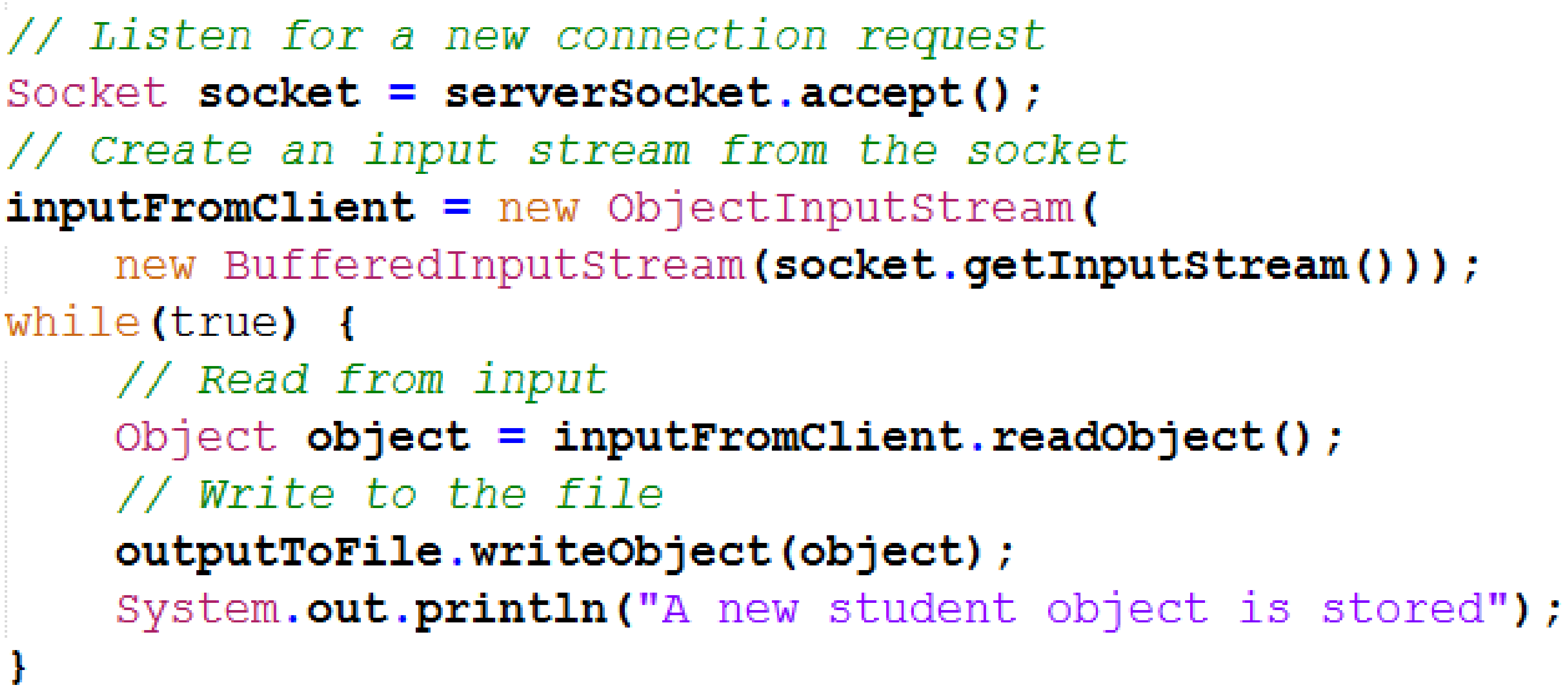
Example: StudentServer.java

- This server will received the object from client and save the object into a file.



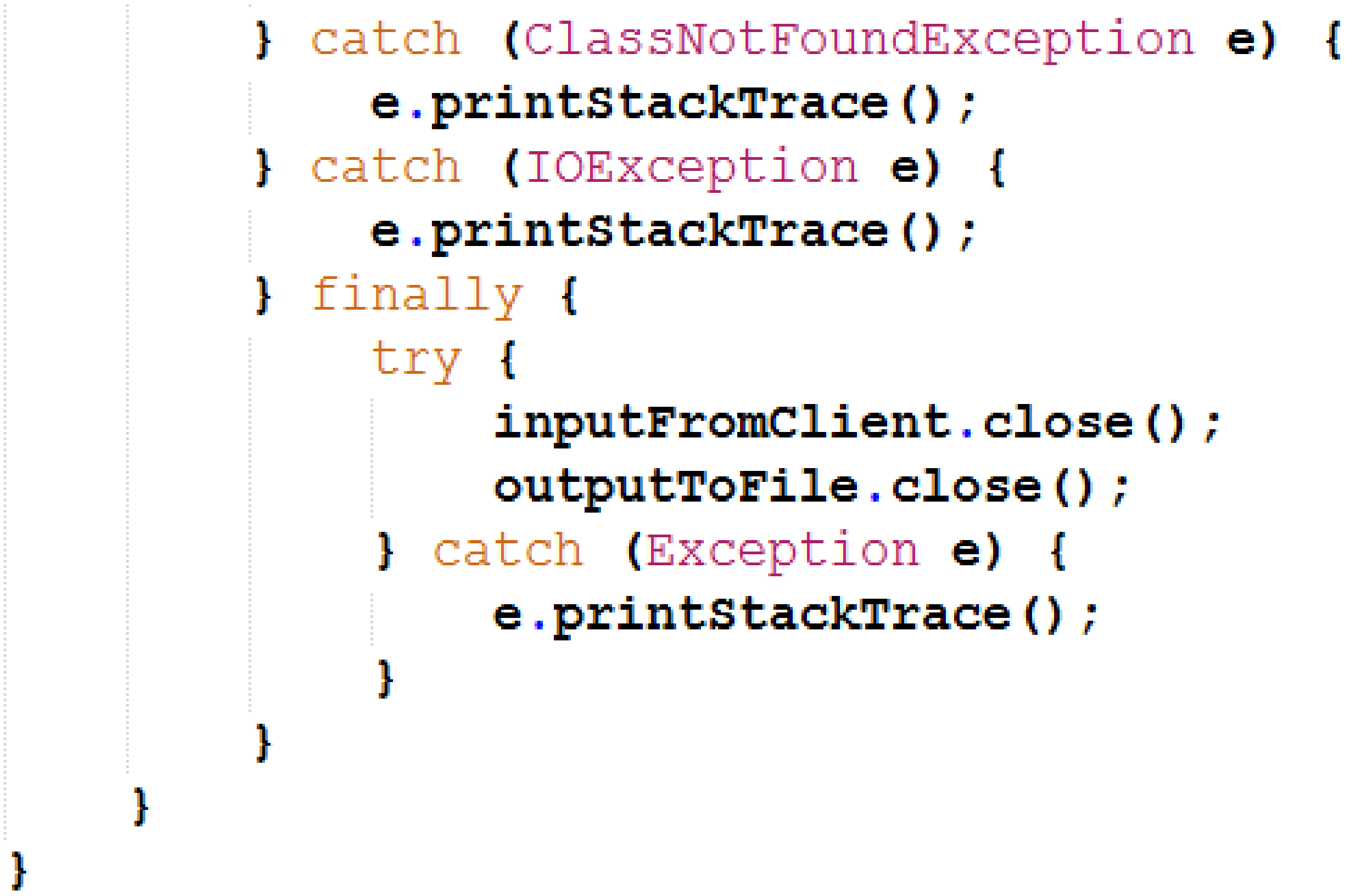
```
public class StudentServer {
    private static ObjectOutputStream outputToFile;
    private static ObjectInputStream inputFromClient;

    public static void main(String[] args) {
        try {
            // Create a server socket
            ServerSocket serverSocket = new ServerSocket(9101);
            System.out.println("Server started");
            // Create an object output stream
            outputToFile = new ObjectOutputStream(
                new BufferedOutputStream(
                    new FileOutputStream("student.dat", true)));
```

Example: StudentServer.java (cont.)



```
// Listen for a new connection request
Socket socket = serverSocket.accept();
// Create an input stream from the socket
inputFromClient = new ObjectInputStream(
    new BufferedInputStream(socket.getInputStream()));
while(true) {
    // Read from input
    Object object = inputFromClient.readObject();
    // Write to the file
    outputToFile.writeObject(object);
    System.out.println("A new student object is stored");
}
```

Example: StudentServer.java (cont.)



```
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                inputFromClient.close();
                outputToFile.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```