

Client-Server communication in C#



AARHUS UNIVERSITY

AARHUS UNIVERSITY SCHOOL OF ENGINEERING

MICHAEL LOFT
ML@ASE.AU.DK



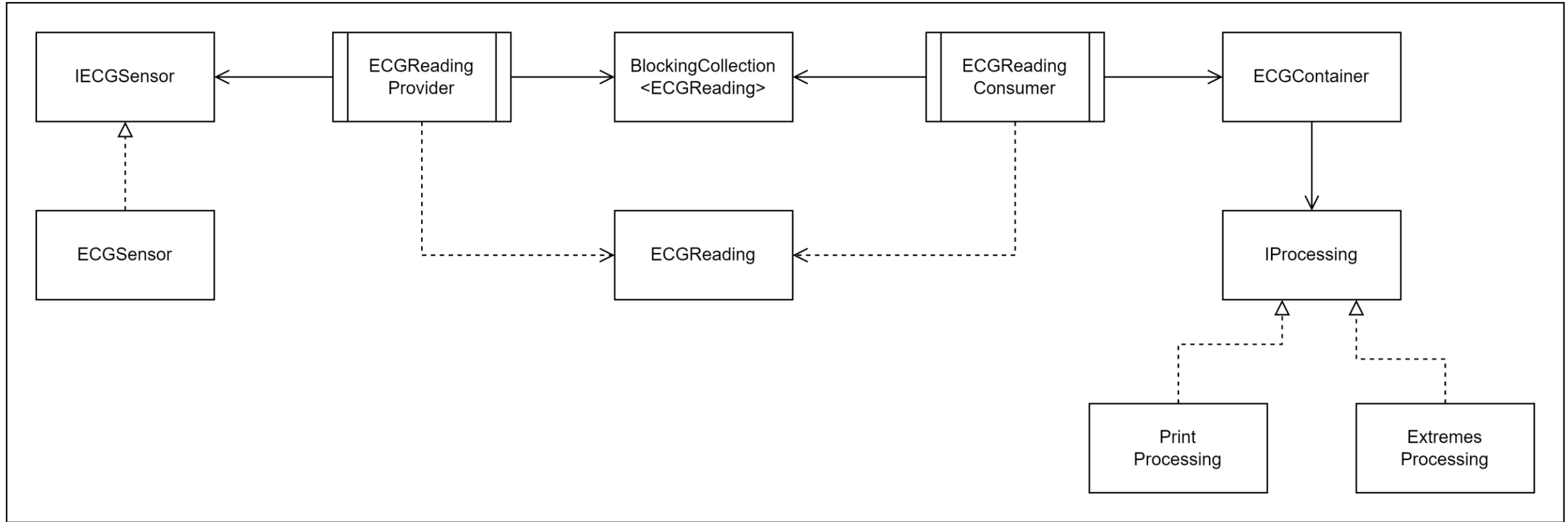
Agenda

The end goal

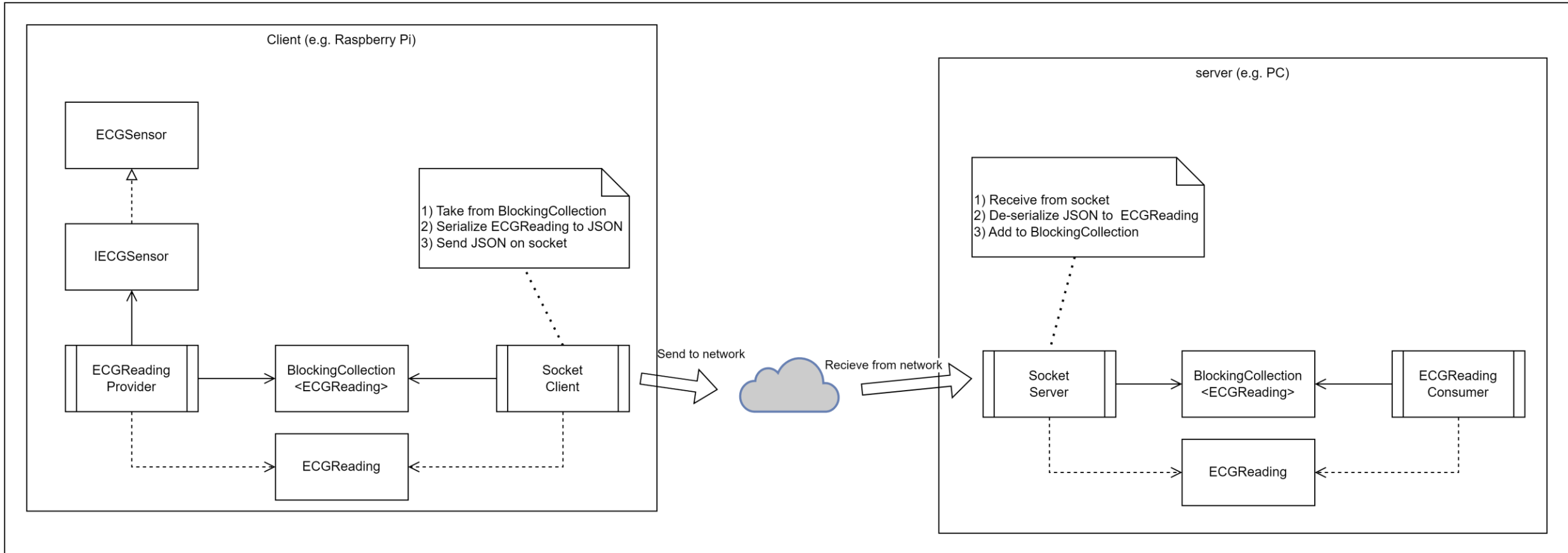
Client <-> Server socket communication

Where we want to end up

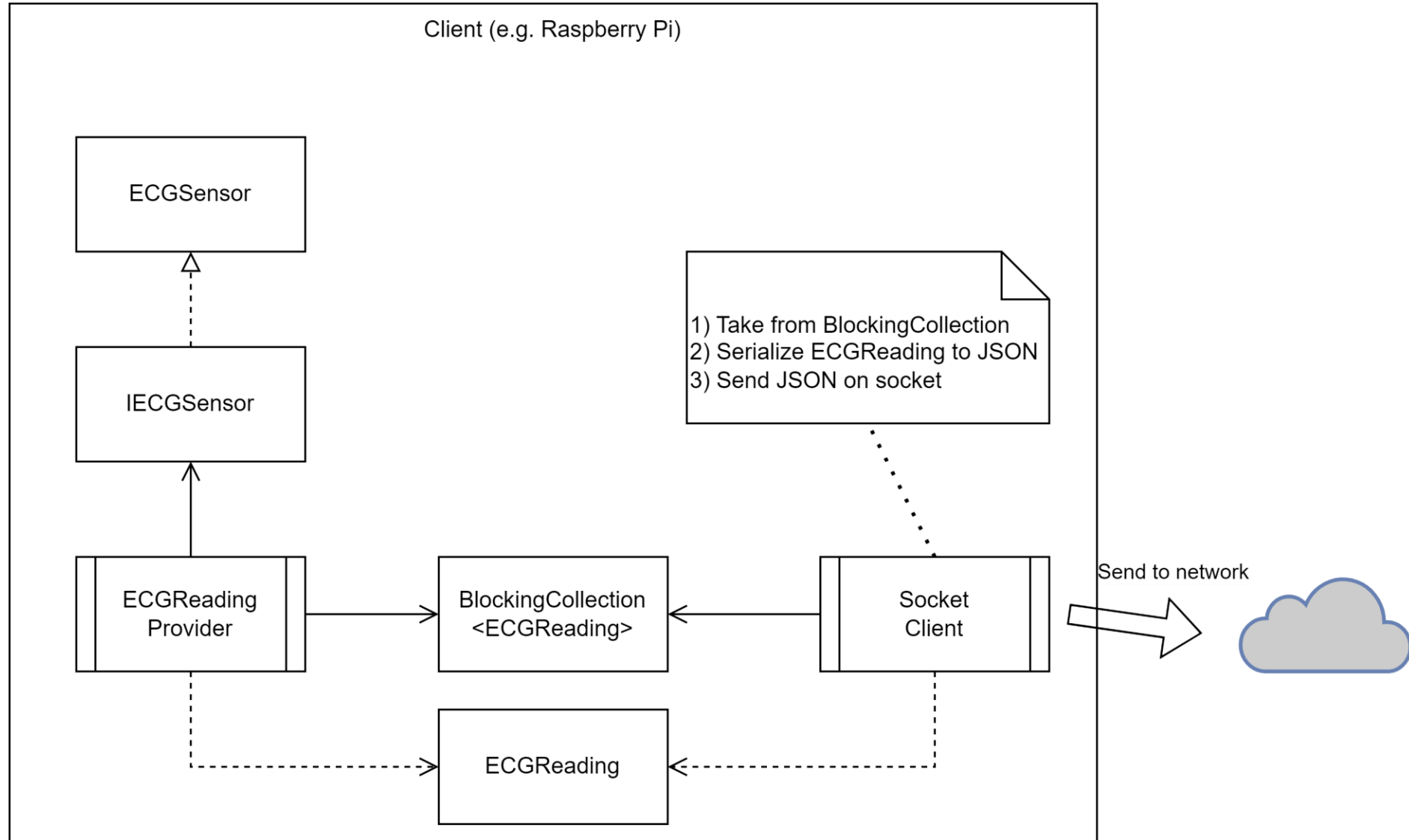
We want to go from this



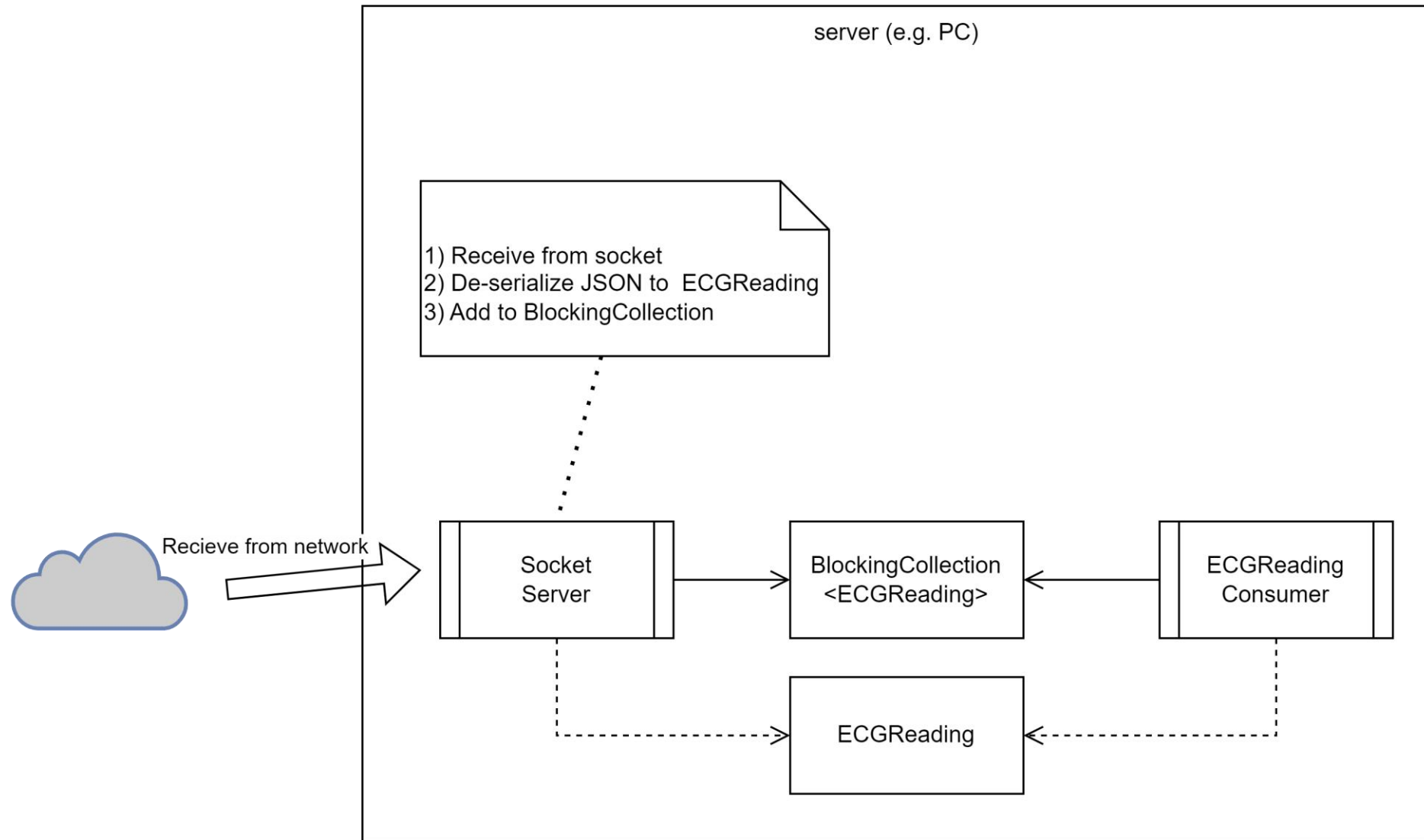
To this



Client

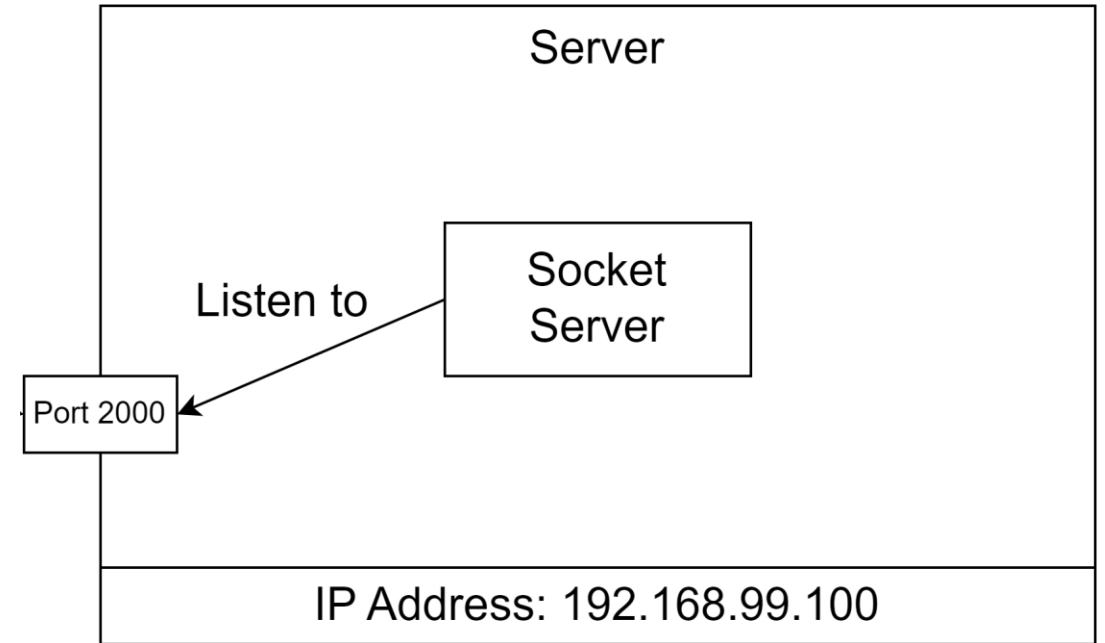


Server



Socket communication

The server listens



The Server specifies where to listen:

- IP Address
- Port number

The server listens

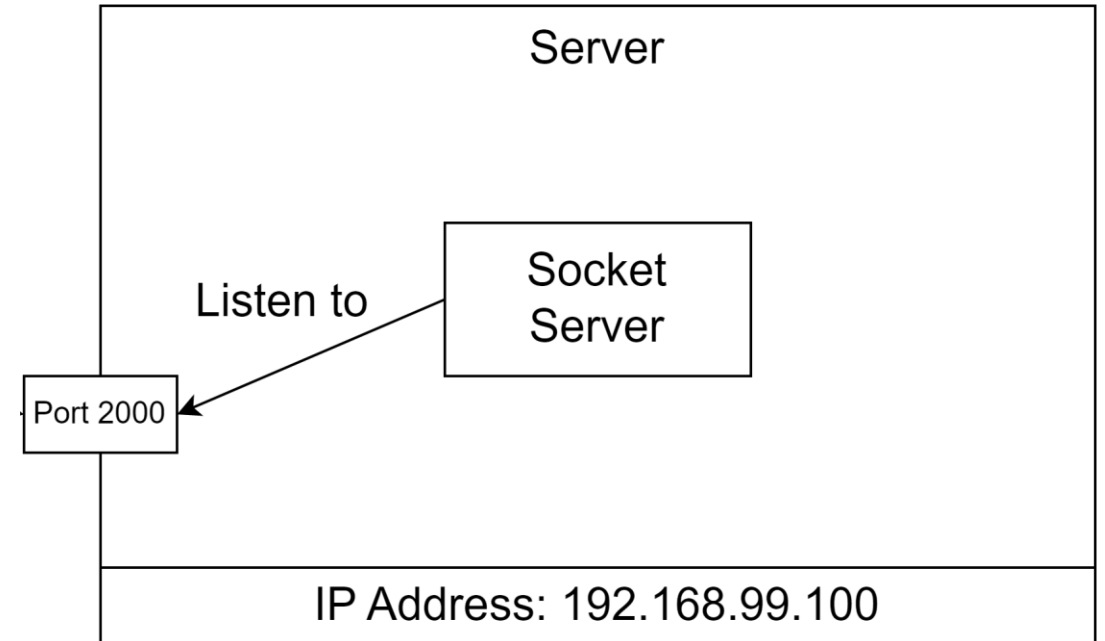
The server can have multiple network cards and more than one address.

IP Address to listen to can be:

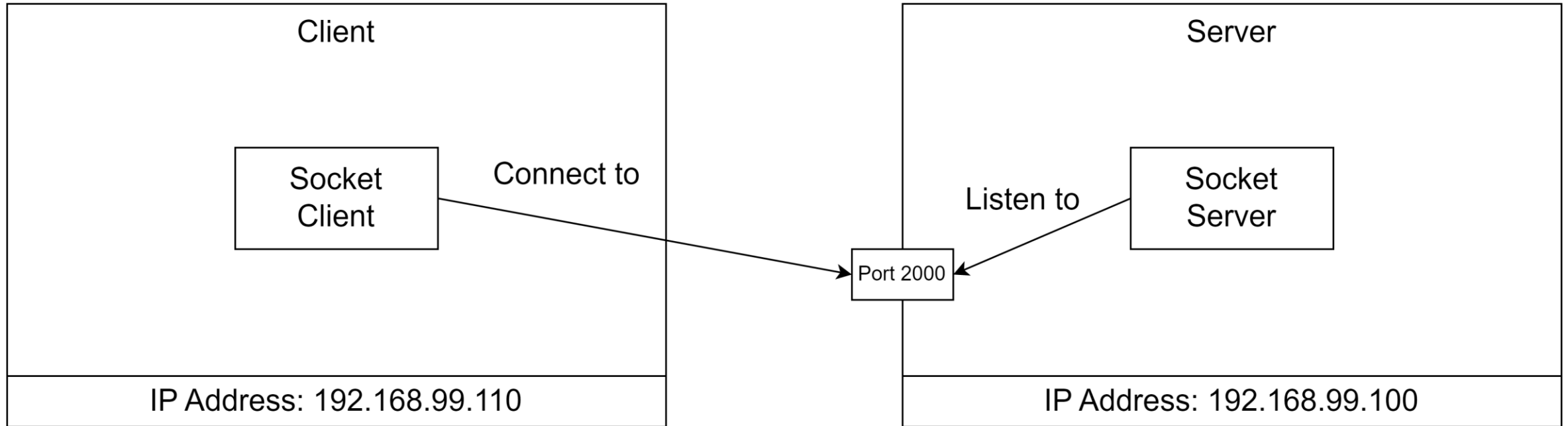
- **Any** \leq all network interfaces
- **Loopback** \leq 127.0.0.1
- A specific IP address

The Server specifies where to listen:

- IP Address
- Port number



The client connects to the server



The client connects to a server socket:

- IP Address
- Port number

SocketServer

```
class SocketServer
{
    public void RunServer()
    {
        // listen to 'Any' which means all network addresses for this machine
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket listener = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        listener.Bind(ipEndPoint);

        Console.WriteLine($"Listening on: {ipAddress}");
        listener.Listen();

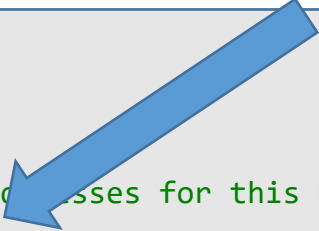
        var handler = listener.Accept();

        while (true)
        {
            byte[] buffer = new byte[1024];
            int numberOfBytesReceived = handler.Receive(buffer, SocketFlags.None);
            string receivedData = Encoding.UTF8.GetString(buffer, 0, numberOfBytesReceived);
            Console.WriteLine($"Server received:{receivedData}");

            string reply = "ACK";
            byte[] replyBytes = Encoding.UTF8.GetBytes(reply);
            handler.Send(replyBytes, SocketFlags.None);
        }
    }
}
```

SocketServer

Create an IPEndPoint with Address = Any and Port = 2000.



```
class SocketServer
{
    public void RunServer()
    {
        // listen to 'Any' which means all network addresses for this machine
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket listener = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        listener.Bind(ipEndPoint);

        Console.WriteLine($"Listening on: {ipAddress}");
        listener.Listen();

        var handler = listener.Accept();

        while (true)
        {
            byte[] buffer = new byte[1024];
            int numberOfBytesReceived = handler.Receive(buffer, SocketFlags.None);
            string receivedData = Encoding.UTF8.GetString(buffer, 0, numberOfBytesReceived);
            Console.WriteLine($"Server received:{receivedData}");

            string reply = "ACK";
            byte[] replyBytes = Encoding.UTF8.GetBytes(reply);
            handler.Send(replyBytes, SocketFlags.None);
        }
    }
}
```

SocketServer

```
class SocketServer
{
    public void RunServer()
    {
        // listen to 'Any' which means all network addresses for this machine
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

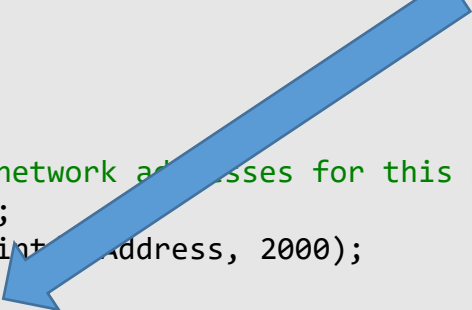
        using Socket listener = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        listener.Bind(ipEndPoint);

        Console.WriteLine($"Listening on: {ipAddress}");
        listener.Listen();

        var handler = listener.Accept();

        while (true)
        {
            byte[] buffer = new byte[1024];
            int numberOfBytesReceived = handler.Receive(buffer, SocketFlags.None);
            string receivedData = Encoding.UTF8.GetString(buffer, 0, numberOfBytesReceived);
            Console.WriteLine($"Server received:{receivedData}");

            string reply = "ACK";
            byte[] replyBytes = Encoding.UTF8.GetBytes(reply);
            handler.Send(replyBytes, SocketFlags.None);
        }
    }
}
```



Create a new socket.

- 'AddressFamily' is the one for IP addresses – "InterNetwork".
- Socket type is Stream, which means a two-way data stream.
- The protocol type is Transmission Control Protocol - TCP.

SocketServer

```
class SocketServer
{
    public void RunServer()
    {
        // listen to 'Any' which means all network addresses for this machine
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket listener = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        listener.Bind(ipEndPoint);

        Console.WriteLine($"Listening on: {ipAddress}");
    }
}
```

Create a new socket.

- 'AddressFamily' is the one for IP addresses – "InterNetwork".
- Socket type is Stream, which means a two-way data stream.
- The protocol type is Transmission Control Protocol - TCP.

The Socket implements the 'IDisposable' interface.

The 'using' keyword means, that the 'listener' object will be disposed correctly, even if any exceptions occur in the code.

```
        handler.Receive(buffer, SocketFlags.None);
        int numberOfBytesReceived = handler.Receive(buffer, 0, numberOfBytesReceived);
        Console.WriteLine($"Received {numberOfBytesReceived} bytes of data");
    }
}
```

```
        byte[] replyBytes = Encoding.UTF8.GetBytes(reply);
        handler.Send(replyBytes, SocketFlags.None);
    }
}
```

SocketServer

Associate socket with local endpoint on the machine.

```
class SocketServer
{
    public void RunServer()
    {
        // listen to 'Any' which means all network addresses for this machine
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket listener = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        listener.Bind(ipEndPoint);

        Console.WriteLine($"Listening on: {ipAddress}");
        listener.Listen();

        var handler = listener.Accept();

        while (true)
        {
            byte[] buffer = new byte[1024];
            int numberOfBytesReceived = handler.Receive(buffer, SocketFlags.None);
            string receivedData = Encoding.UTF8.GetString(buffer, 0, numberOfBytesReceived);
            Console.WriteLine($"Server received:{receivedData}");

            string reply = "ACK";
            byte[] replyBytes = Encoding.UTF8.GetBytes(reply);
            handler.Send(replyBytes, SocketFlags.None);
        }
    }
}
```


SocketServer

Start listening on the socket.

```
class SocketServer
{
    public void RunServer()
    {
        // listen to 'Any' which means all network addresses for this machine
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

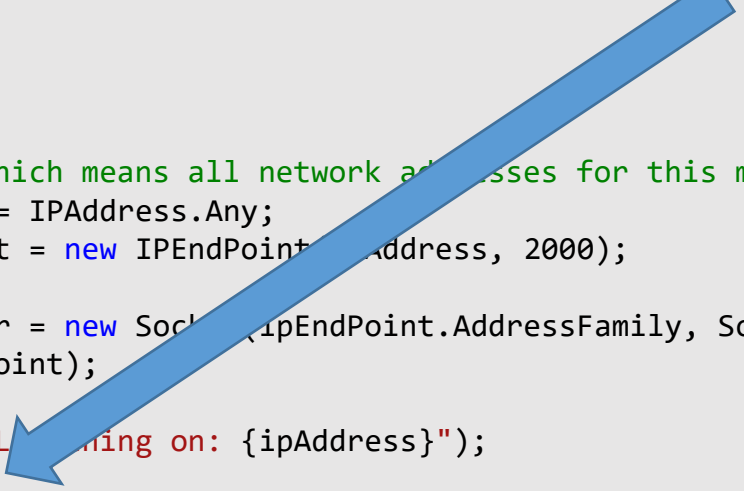
        using Socket listener = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        listener.Bind(ipEndPoint);

        Console.WriteLine($"Listening on: {ipAddress}");
        listener.Listen();

        var handler = listener.Accept();

        while (true)
        {
            byte[] buffer = new byte[1024];
            int numberOfBytesReceived = handler.Receive(buffer, SocketFlags.None);
            string receivedData = Encoding.UTF8.GetString(buffer, 0, numberOfBytesReceived);
            Console.WriteLine($"Server received:{receivedData}");

            string reply = "ACK";
            byte[] replyBytes = Encoding.UTF8.GetBytes(reply);
            handler.Send(replyBytes, SocketFlags.None);
        }
    }
}
```



SocketServer

Wait for a connection from a client.

```
class SocketServer
{
    public void RunServer()
    {
        // listen to 'Any' which means all network addresses on this machine
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 8080);

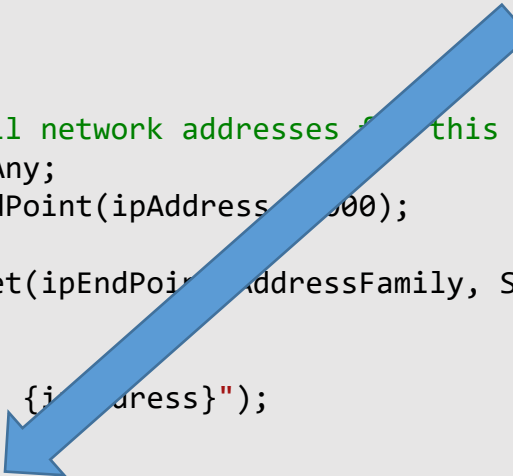
        using Socket listener = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        listener.Bind(ipEndPoint);

        Console.WriteLine($"Listening on: {ipEndPoint.Address}");
        listener.Listen();

        var handler = listener.Accept();

        while (true)
        {
            byte[] buffer = new byte[1024];
            int numberOfBytesReceived = handler.Receive(buffer, SocketFlags.None);
            string receivedData = Encoding.UTF8.GetString(buffer, 0, numberOfBytesReceived);
            Console.WriteLine($"Server received:{receivedData}");

            string reply = "ACK";
            byte[] replyBytes = Encoding.UTF8.GetBytes(reply);
            handler.Send(replyBytes, SocketFlags.None);
        }
    }
}
```



SocketServer

```
class SocketServer
{
    public void RunServer()
    {
        // listen to 'Any' which means all network addresses for this machine
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket listener = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        listener.Bind(ipEndPoint);


        Console.WriteLine($"Listening on: {ipAddress}");
        listener.Listen();

        var handler = listener.Accept();

        while (true)
        {
            byte[] buffer = new byte[1024];
            int numberOfBytesReceived = handler.Receive(buffer, SocketFlags.None);
            string receivedData = Encoding.UTF8.GetString(buffer, 0, numberOfBytesReceived);
            Console.WriteLine($"Server received:{receivedData}");

            string reply = "ACK";
            byte[] replyBytes = Encoding.UTF8.GetBytes(reply);
            handler.Send(replyBytes, SocketFlags.None);
        }
    }
}
```

Read bytes from the socket in to a buffer



SocketServer

```
class SocketServer
{
    public void RunServer()
    {
        // listen to 'Any' which means all network addresses for this machine
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket listener = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        listener.Bind(ipEndPoint);

        Console.WriteLine($"Listening on: {ipAddress}");
        listener.Listen();

        var handler = listener.Accept();

        while (true)
        {
            byte[] buffer = new byte[1024];
            int numberOfBytesReceived = handler.Receive(buffer, SocketFlags.None);
            string receivedData = Encoding.UTF8.GetString(buffer, 0, numberOfBytesReceived);
            Console.WriteLine($"Server received:{receivedData}");

            string reply = "ACK";
            byte[] replyBytes = Encoding.UTF8.GetBytes(reply);
            handler.Send(replyBytes, SocketFlags.None);
        }
    }
}
```

Convert to text (UTF-8)



SocketServer

```
class SocketServer
{
    public void RunServer()
    {
        // listen to 'Any' which means all network addresses for this machine
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket listener = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        listener.Bind(ipEndPoint);

        Console.WriteLine($"Listening on: {ipAddress}");
        listener.Listen();

        var handler = listener.Accept();

        while (true)
        {
            byte[] buffer = new byte[1024];
            int numberOfBytesReceived = handler.Receive(buffer, SocketFlags.None);
            string receivedData = Encoding.UTF8.GetString(buffer, 0, numberOfBytesReceived);
            Console.WriteLine($"Server received:{receivedData}");

            string reply = "ACK";
            byte[] replyBytes = Encoding.UTF8.GetBytes(reply);
            handler.Send(replyBytes, SocketFlags.None);
        }
    }
}
```

Read bytes from the socket in to a buffer



SocketServer

```
class SocketServer
{
    public void RunServer()
    {
        // listen to 'Any' which means all network addresses for this machine
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket listener = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        listener.Bind(ipEndPoint);

        Console.WriteLine($"Listening on: {ipAddress}");
        listener.Listen();

        var handler = listener.Accept();

        while (true)
        {
            byte[] buffer = new byte[1024];
            int numberOfBytesReceived = handler.Receive(buffer, SocketFlags.None);
            string receivedData = Encoding.UTF8.GetString(buffer, 0, numberOfBytesReceived);
            Console.WriteLine($"Server received:{receivedData}");

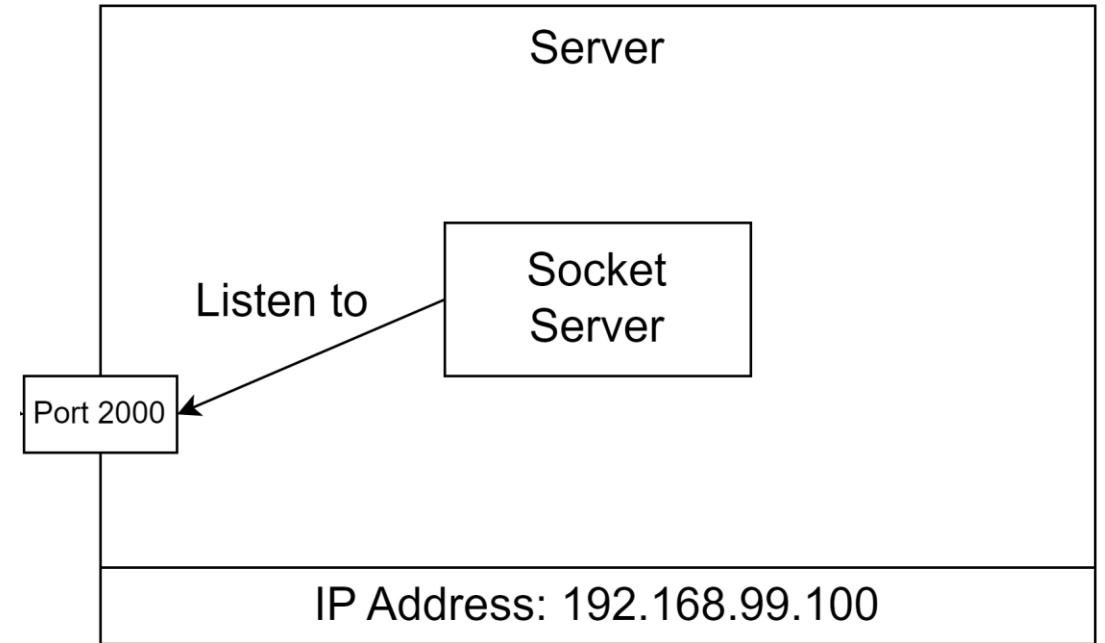
            string reply = "ACK";
            byte[] replyBytes = Encoding.UTF8.GetBytes(reply);
            handler.Send(replyBytes, SocketFlags.None);
        }
    }
}
```

Send "ACK" message back to the client.

NOTE:

There is no requirement to send anything back to the client. It is only included here to show two-way communication.

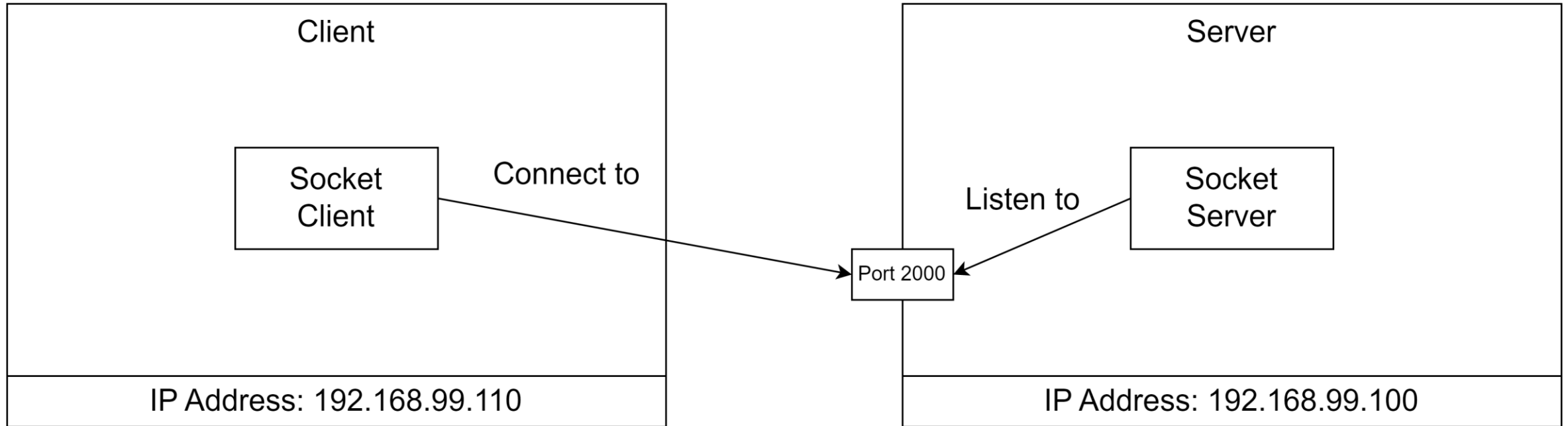
The server listens



The Server specifies where to listen:

- IP Address = Any
- Port number = 2000

The client connects to the server



The client connects to a server socket:

- IP Address
- Port number

SocketClient

```
class SocketClient
{
    public void RunClient()
    {
        IPAddress ipAddress = IPAddress.Parse("127.0.0.1");
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket client = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        client.Connect(ipEndPoint);

        for (int i = 0; i < 10; i++)
        {
            // Send message.
            var message = "Hello " + i;
            var messageBytes = Encoding.UTF8.GetBytes(message);
            client.Send(messageBytes, SocketFlags.None);
            Console.WriteLine($"Socket client sent message: {message}");

            // Receive ack.
            var buffer = new byte[1024];
            var received = client.Receive(buffer, SocketFlags.None);
            var response = Encoding.UTF8.GetString(buffer, 0, received);

            Console.WriteLine($"Client received: {response}");
            Thread.Sleep(1000);
        }

        client.Shutdown(SocketShutdown.Both);
    }
}
```

SocketClient

```
class SocketClient
{
    public void RunClient()
    {
        IPAddress ipAddress = IPAddress.Parse("127.0.0.1");
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket client = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        client.Connect(ipEndPoint);

        for (int i = 0; i < 10; i++)
        {
            // Send message.
            var message = "Hello " + i;
            var messageBytes = Encoding.UTF8.GetBytes(message);
            client.Send(messageBytes, SocketFlags.None);
            Console.WriteLine($"Socket client sent message: {message}");

            // Receive ack.
            var buffer = new byte[1024];
            var received = client.Receive(buffer, SocketFlags.None);
            var response = Encoding.UTF8.GetString(buffer, 0, received);

            Console.WriteLine($"Client received: {response}");
            Thread.Sleep(1000);
        }

        client.Shutdown(SocketShutdown.Both);
    }
}
```

Create an IPEndPoint with
Address = 127.0.0.1 (localhost)
and
Port = 2000.

If the server is on a different machine, the
Address shall be the IP address of that machine.

SocketClient

Create the socket and connect to the IP Endpoint.

The connection attempt may time out if the endpoint does not exist, which will throw an exception.

```
class SocketClient
{
    public void RunClient()
    {
        IPAddress ipAddress = IPAddress.Parse("127.0.0.1");
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket client = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        client.Connect(ipEndPoint);

        for (int i = 0; i < 10; i++)
        {
            // Send message.
            var message = "Hello " + i;
            var messageBytes = Encoding.UTF8.GetBytes(message);
            client.Send(messageBytes, SocketFlags.None);
            Console.WriteLine($"Socket client sent message: {message}");

            // Receive ack.
            var buffer = new byte[1024];
            var received = client.Receive(buffer, SocketFlags.None);
            var response = Encoding.UTF8.GetString(buffer, 0, received);

            Console.WriteLine($"Client received: {response}");
            Thread.Sleep(1000);
        }

        client.Shutdown(SocketShutdown.Both);
    }
}
```



SocketClient

Create the message to send.
Convert to bytes.
Send the bytes.

```
class SocketClient
{
    public void RunClient()
    {
        IPAddress ipAddress = IPAddress.Parse("127.0.0.1");
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket client = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        client.Connect(ipEndPoint);

        for (int i = 0; i < 10; i++)
        {
            // Send message.
            var message = "Hello " + i;
            var messageBytes = Encoding.UTF8.GetBytes(message);
            client.Send(messageBytes, SocketFlags.None);
            Console.WriteLine($"Socket client sent message: {message}");

            // Receive ack.
            var buffer = new byte[1024];
            var received = client.Receive(buffer, SocketFlags.None);
            var response = Encoding.UTF8.GetString(buffer, 0, received);

            Console.WriteLine($"Client received: {response}");
            Thread.Sleep(1000);
        }

        client.Shutdown(SocketShutdown.Both);
    }
}
```



SocketClient

Receive "ACK" message from the server.

NOTE:

There is no requirement to send anything back to the client. It is only included here to show two-way communication.

```
class SocketClient
{
    public void RunClient()
    {
        IPAddress ipAddress = IPAddress.Parse("127.0.0.1");
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);


        using Socket client = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        client.Connect(ipEndPoint);

        for (int i = 0; i < 10; i++)
        {
            // Send message.
            var message = "Hello " + i;
            var messageBytes = Encoding.UTF8.GetBytes(message);
            client.Send(messageBytes, SocketFlags.None);
            Console.WriteLine($"Socket client sent message: {message}");

            // Receive ack.
            var buffer = new byte[1024];
            var received = client.Receive(buffer, SocketFlags.None);
            var response = Encoding.UTF8.GetString(buffer, 0, received);

            Console.WriteLine($"Client received: {response}");
            Thread.Sleep(1000);
        }

        client.Shutdown(SocketShutdown.Both);
    }
}
```



SocketClient

Shutdown both sending and receiving socket when done.

```
class SocketClient
{
    public void RunClient()
    {
        IPAddress ipAddress = IPAddress.Parse("127.0.0.1");
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

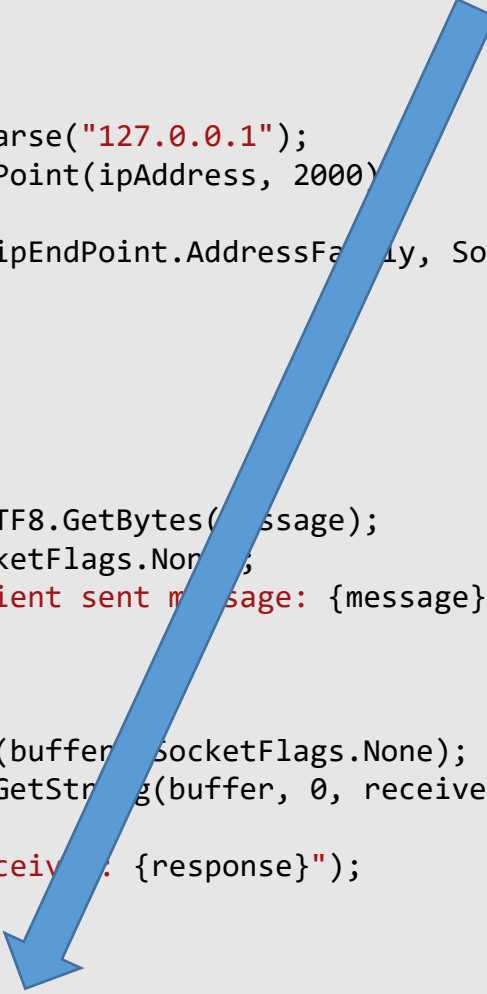
        using Socket client = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        client.Connect(ipEndPoint);

        for (int i = 0; i < 10; i++)
        {
            // Send message.
            var message = "Hello " + i;
            var messageBytes = Encoding.UTF8.GetBytes(message);
            client.Send(messageBytes, SocketFlags.None);
            Console.WriteLine($"Socket client sent message: {message}");

            // Receive ack.
            var buffer = new byte[1024];
            var received = client.Receive(buffer, SocketFlags.None);
            var response = Encoding.UTF8.GetString(buffer, 0, received);

            Console.WriteLine($"Client received: {response}");
            Thread.Sleep(1000);
        }

        client.Shutdown(SocketShutdown.Both);
    }
}
```





Your turn

Solve the
”Exercise - Client-Server
communication”

Sending objects from the client

```
internal class SocketClient
{
    private readonly BlockingCollection<ECGReading> _ecgReadings;

    public SocketClient(BlockingCollection<ECGReading> ecgReadings)
    {
        _ecgReadings = ecgReadings;
    }

    public void Run()
    {
        IPAddress ipAddress = IPAddress.Parse("127.0.0.1");
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket client = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        client.Connect(ipEndPoint);

        while (!_ecgReadings.IsCompleted)
        {
            try
            {
                ECGReading ecgReading = _ecgReadings.Take();

                string objectAsJson = JsonSerializer.Serialize(ecgReading);
                var messageBytes = Encoding.UTF8.GetBytes(objectAsJson);
                client.Send(messageBytes, SocketFlags.None);
                Console.WriteLine($"Socket client sent message: {objectAsJson}");
            }
            catch (InvalidOperationException)
            {
                // IOE means that Take() was called on a completed collection.
            }
        }
        client.Shutdown(SocketShutdown.Both);
    }
}
```


Sending objects from the client

```
internal class SocketClient
{
    private readonly BlockingCollection<ECGReading> _ecgReadings;

    public SocketClient(BlockingCollection<ECGReading> ecgReadings)
    {
        _ecgReadings = ecgReadings;
    }

    public void Run()
    {
        IPAddress ipAddress = IPAddress.Parse("127.0.0.1");
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket client = new Socket(ipEndPoint.AddressFamily,
SocketType.Stream, ProtocolType.Tcp);
        client.Connect(ipEndPoint);
    }
}
```

```
client.Connect(ipEndPoint);

while (!_ecgReadings.IsCompleted)
{
    try
    {
        ECGReading ecgReading = _ecgReadings.Take();

        string objectAsJson = JsonSerializer.Serialize(ecgReading);
        var messageBytes = Encoding.UTF8.GetBytes(objectAsJson);
        client.Send(messageBytes, SocketFlags.None);
        Console.WriteLine($"Socket client sent message: {objectAsJson}");
    }
    catch (InvalidOperationException)
    {
        // IOE means that Take() was called on a completed collection.
    }
}
client.Shutdown(SocketShutdown.Both);
}
```

Receiving objects on the server

```
internal class SocketServer
{
    private readonly BlockingCollection<ECGReading> _ecgReadings;

    public SocketServer(BlockingCollection<ECGReading> ecgReadings)
    {
        _ecgReadings = ecgReadings;
    }

    public void Run()
    {
        RunServer();
    }

    public void RunServer()
    {
        // listen to 'Any' which means all network addresses for this machine
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket listener = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);

        listener.Bind(ipEndPoint);

        Console.WriteLine($"Listening on: {ipAddress}");
        listener.Listen();

        var handler = listener.Accept();

        while (!ShallStop)
        {
            byte[] buffer = new byte[1024];
            int numberOfBytesReceived = handler.Receive(buffer, SocketFlags.None);
            if (numberOfBytesReceived > 0)
            {
                string receivedData = Encoding.UTF8.GetString(buffer, 0, numberOfBytesReceived);
                Console.WriteLine($"Server received: {receivedData}");

                try
                {
                    ECGReading? ecgReading = JsonSerializer.Deserialize<ECGReading>(receivedData);
                    if (ecgReading != null) _ecgReadings.Add(ecgReading);
                }
                catch (System.Text.Json.JsonException e)
                {
                    // log any parsing exceptions
                    Console.WriteLine(e);
                }
            }
        }
        listener.Close();
        _ecgReadings.CompleteAdding();
    }
}
```

```
public bool ShallStop { get; set; } = false;
```

```
}
```

Receiving objects on the server

```
internal class SocketServer
{
    private readonly BlockingCollection<ECGReading> _ecgReadings;

    public SocketServer(BlockingCollection<ECGReading> ecgReadings)
    {
        _ecgReadings = ecgReadings;
    }

    public void Run()
    {
        RunServer();
    }

    public void RunServer()
    {
        // listen to 'Any' which means all network addresses for this machine
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket listener = new Socket(ipEndPoint.AddressFamily,
```

```
using Socket listener = new Socket(ipEndPoint.AddressFamily,
SocketType.Stream, ProtocolType.Tcp);

listener.Bind(ipEndPoint);

Console.WriteLine($"Listening on: {ipAddress}");
listener.Listen();

var handler = listener.Accept();

while (!ShallStop)
{
    byte[] buffer = new byte[1024];
    int numberOfBytesReceived = handler.Receive(buffer, SocketFlags.None);
    if (numberOfBytesReceived > 0)
    {
        string receivedData = Encoding.UTF8.GetString(buffer, 0,
numberOfBytesReceived);
        Console.WriteLine($"Server received:{receivedData}");

        try
        {
            ECGReading? ecgReading =
JsonSerializer.Deserialize<ECGReading>(receivedData):
```

```

        {
            string receivedData = Encoding.UTF8.GetString(buffer, 0,
numberOfBytesReceived);
            Console.WriteLine($"Server received:{receivedData}");

            try
            {
                ECGReading? ecgReading =
JsonSerializer.Deserialize<ECGReading>(receivedData);
                if (ecgReading != null) _ecgReadings.Add(ecgReading);
            }
            catch (System.Text.Json.JsonException e)
            {
                // log any parsing exceptions
                Console.WriteLine(e);
            }
        }
    }
    listener.Close();
    _ecgReadings.CompleteAdding();
}

public bool ShallStop { get; set; } = false;
}

```



Your turn

Solve the
” Exercise - ECG Network”

References and image sources

Computer keyboard:

http://stockmedia.cc/computing_technology/slides/DSD_8790.jpg

Warning tape: <https://www.pngall.com/warning-sign-png/download/69428>



AARHUS UNIVERSITY