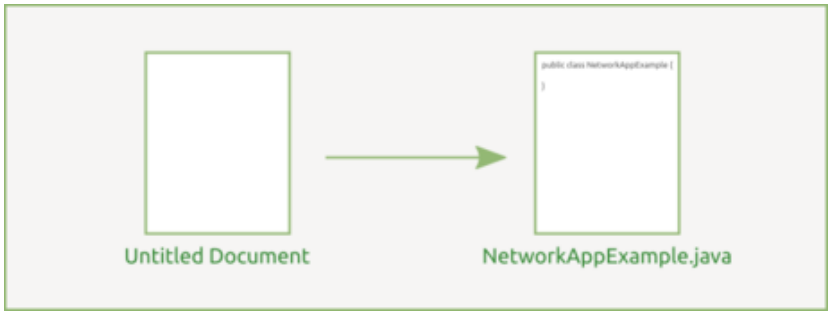
How to Create a Network Application in Java

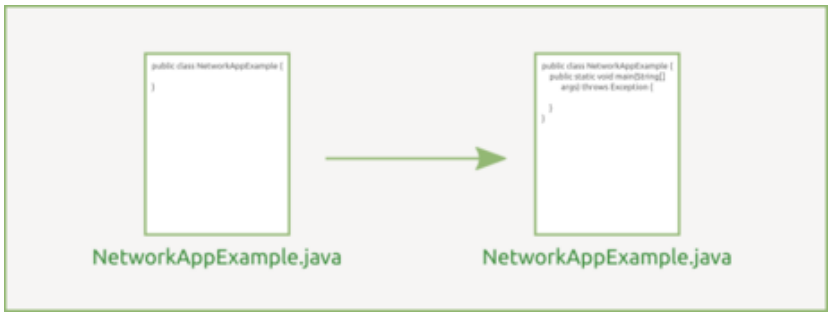
Writing code that executes on a certain device is very satisfying. But, writing code that executes on several devices that communicate with each other is simply life-affirming. This article will teach you how to connect and exchange...

Steps

1. 

Create a class. Create a class and name it however you want. In this article, it will be named `NetworkAppExample`.

```
public class NetworkAppExample { }
```

2. 

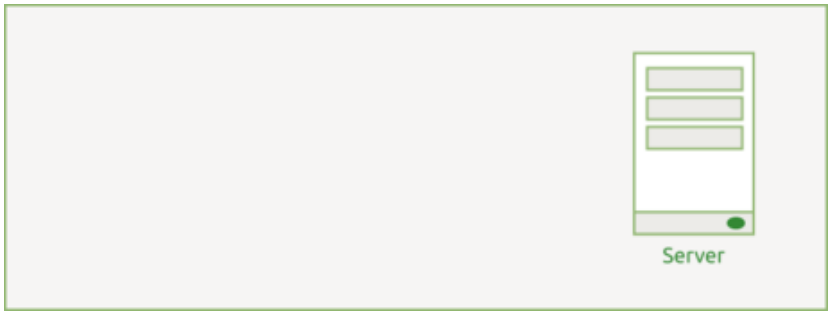
Create a main method. Create a main method and declare it might throw exceptions of `Exception` type and any subclass of it - all exceptions. This is considered a bad practice, but is acceptable for barebone examples.

```
public class NetworkAppExample { public static void main(String[] args)
 throws Exception { } }
```

3. 

Declare server address. This example will use local host address and an arbitrary port number. Port number needs to be in a range from 0 to 65535 (inclusive). However, port numbers to avoid range from 0 to 1023 (inclusive) because they are reserved system ports.

```
public class NetworkAppExample { public static void main(String[] args)
 throws Exception { String host = "localhost"; int port = 10430; } }
```

4. 

Create a server. Server is bound to the address and port and listens for incoming connections. In Java, `ServerSocket` represents server-side endpoint and its function is accepting new connections. `ServerSocket` does not have streams for reading and sending data because it does not represent connection between a server and a client.

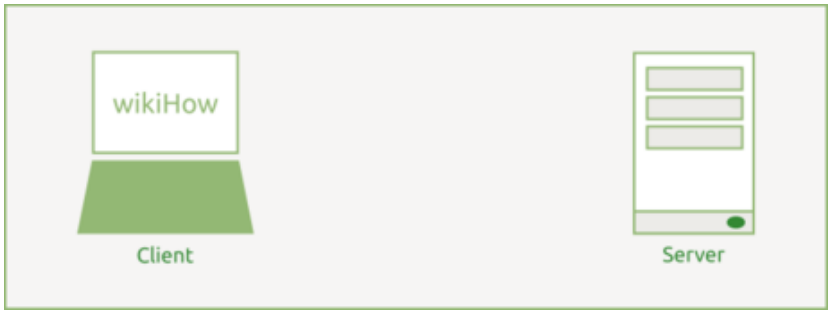
```
import java.net.InetAddress; import java.net.ServerSocket; public class
 NetworkAppExample { public static void main(String[] args) throws
Exception { String host = "localhost"; int port = 10430; ServerSocket
server = new ServerSocket(port, 50, InetAddress.getByName(host)); } }
```

5. 

Log server inception. For logging purposes, print to the console that server has been started.

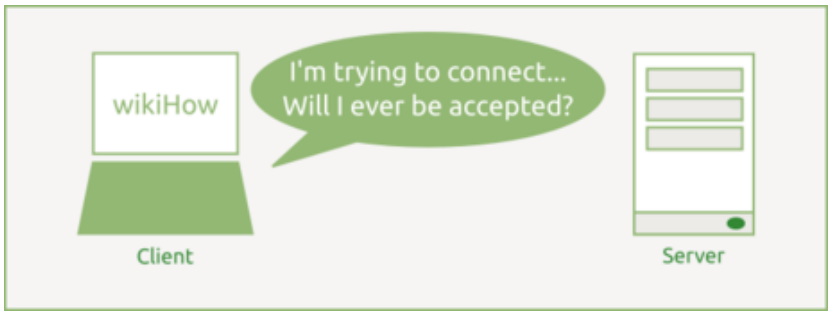
```
import java.net.InetAddress; import java.net.ServerSocket; public class
 NetworkAppExample { public static void main(String[] args) throws
```

```
Exception { String host = "localhost"; int port = 10430; ServerSocket
server = new ServerSocket(port, 50, InetAddress.getByName(host));
System.out.println("Server started."); } }
```

6. 

Create a client. Client is bound to the address and port of a server and listens for packets (messages) after connection is established. In Java, `Socket` represents either a client-side endpoint connected to the server or a connection (from server) to client and is used to communicate with the party on the other end.

```
import java.net.InetAddress; import java.net.ServerSocket; import
java.net.Socket; public class NetworkAppExample { public static void
main(String[] args) throws Exception { String host = "localhost"; int
port = 10430; ServerSocket server = new ServerSocket(port, 50,
InetAddress.getByName(host)); System.out.println("Server started.");
Socket client = new Socket(host, port); } }
```

7. 

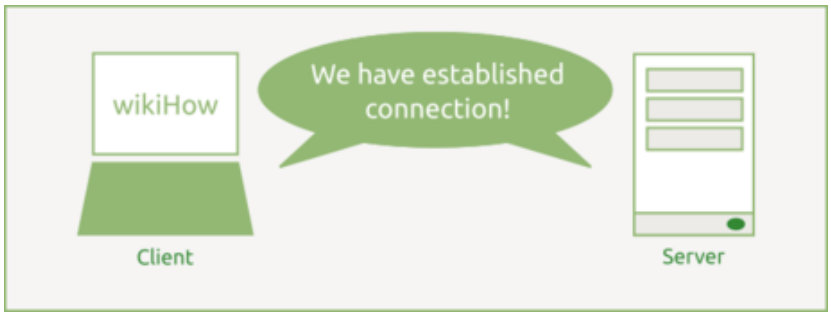
Log connection attempt. For logging purposes, print to the console that connection has been attempted.

```
import java.net.InetAddress; import java.net.ServerSocket; import
java.net.Socket; public class NetworkAppExample { public static void
main(String[] args) throws Exception { String host = "localhost"; int
port = 10430; ServerSocket server = new ServerSocket(port, 50,
InetAddress.getByName(host)); System.out.println("Server started.");
Socket client = new Socket(host, port); System.out.println(
"Connecting to server..."); } }
```

8. 

Establish connection. Clients will never connect unless server listens for and accepts, in other words establishes, connections. In Java, connections are established using `accept()` method of `ServerSocket` class. The method will block execution until a client connects.

```
import java.net.InetAddress; import java.net.ServerSocket; import
java.net.Socket; public class NetworkAppExample { public static void
main(String[] args) throws Exception { String host = "localhost"; int
port = 10430; ServerSocket server = new ServerSocket(port, 50,
InetAddress.getByName(host)); System.out.println("Server started.");
Socket client = new Socket(host, port); System.out.println(
"Connecting to server..."); Socket connection = server.accept(); } }
```

9. 

Log established connection. For logging purposes, print to the console that connection between server and client has been established.

```
import java.net.InetAddress; import java.net.ServerSocket; import
java.net.Socket; public class NetworkAppExample { public static void
main(String[] args) throws Exception { String host = "localhost"; int
port = 10430; ServerSocket server = new ServerSocket(port, 50,
InetAddress.getByName(host)); System.out.println("Server started.");
Socket client = new Socket(host, port); System.out.println(
"Connecting to server..."); Socket connection = server.accept(); System
.out.println("Connection established."); } }
```

10. 

Prepare communication streams. Communication is done over streams and, in this application, raw streams of (connection from) server (to client) and client need to be chained to either data or object streams. Remember, both parties need to use the same stream type.

1. **Data streams**

```
import java.io.DataInputStream; import java.io.DataOutputStream;
import java.net.InetAddress; import java.net.ServerSocket; import
java.net.Socket; public class NetworkAppExample { public static
void main(String[] args) throws Exception { String host =
"localhost"; int port = 10430; ServerSocket server = new
ServerSocket(port, 50, InetAddress.getByName(host)); System.out.
println("Server started."); Socket client = new Socket(host, port);
 System.out.println("Connecting to server..."); Socket connection =
 server.accept(); System.out.println("Connection established.");
DataOutputStream clientOut = new DataOutputStream(client.
getOutputStream()); DataInputStream clientIn = new DataInputStream(
client.getInputStream()); DataOutputStream serverOut = new
DataOutputStream(connection.getOutputStream()); DataInputStream
serverIn = new DataInputStream(connection.getInputStream()); } }
```

2. **Object streams**

   When multiple object streams are used, input streams have to be initialized in the same order as output streams because `ObjectOutputStream` sends a header to the other party and `ObjectInputStream` blocks execution until it reads the header.

```
import java.io.ObjectInputStream; import java.io.ObjectOutputStream
; import java.net.InetAddress; import java.net.ServerSocket; import
 java.net.Socket; public class NetworkAppExample { public static
void main(String[] args) throws Exception { String host =
"localhost"; int port = 10430; ServerSocket server = new
ServerSocket(port, 50, InetAddress.getByName(host)); System.out.
println("Server started."); Socket client = new Socket(host, port);
 System.out.println("Connecting to server..."); Socket connection =
 server.accept(); System.out.println("Connection established.");
ObjectOutputStream clientOut = new ObjectOutputStream(client.
getOutputStream()); ObjectOutputStream serverOut = new
ObjectOutputStream(connection.getOutputStream()); ObjectInputStream
 clientIn = new ObjectInputStream(client.getInputStream());
ObjectInputStream serverIn = new ObjectInputStream(connection.
getInputStream()); } }
```

Order as specified in the code above might be easier to remember - first initialize output streams then input streams in the same order. However, another order for initialization of object streams is the following:

```
ObjectOutputStream clientOut = new ObjectOutputStream(client.
getOutputStream()); ObjectInputStream serverIn = new
ObjectInputStream(connection.getInputStream()); ObjectOutputStream
serverOut = new ObjectOutputStream(connection.getOutputStream());
ObjectInputStream clientIn = new ObjectInputStream(client.
getInputStream());
```

11. 

Log that communication is ready. For logging purposes, print to the console that communication is ready.

```
// code omitted import java.net.InetAddress; import
java.net.ServerSocket; import java.net.Socket; public class
NetworkAppExample { public static void main(String[] args) throws
Exception { String host = "localhost"; int port = 10430; ServerSocket
server = new ServerSocket(port, 50, InetAddress.getByName(host));
System.out.println("Server started."); Socket client = new Socket(host,
 port); System.out.println("Connecting to server..."); Socket
connection = server.accept(); System.out.println(
"Connection established."); // code omitted System.out.println(
"Communication is ready."); } }
```

12. 

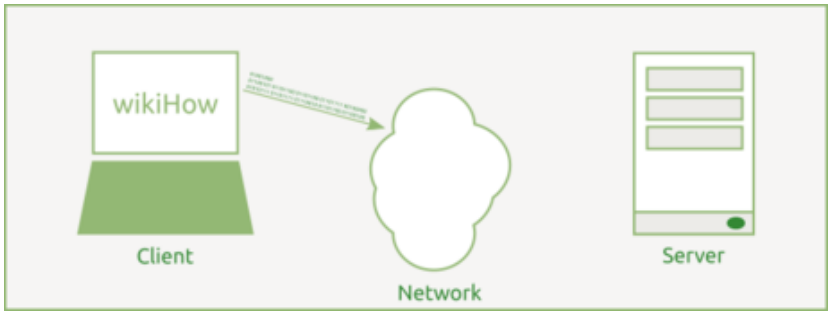
Create a message. In this application, `Hello World` text will be sent to the server either as `byte[]` or `String`. Declare a variable of the type that depends on the stream used. Use `byte[]` for data streams and `String` for object streams.

1. **Data streams**
   Using data streams, serialization is done by converting objects into primitive data types or a `String`. In this case, `String` is converted to `byte[]` instead of written using `writeBytes()` method to show how it would be done with other objects, such as images or other files.

```
import java.io.DataInputStream; import java.io.DataOutputStream;
import java.net.InetAddress; import java.net.ServerSocket; import
java.net.Socket; public class NetworkAppExample { public static
void main(String[] args) throws Exception { String host =
"localhost"; int port = 10430; ServerSocket server = new
ServerSocket(port, 50, InetAddress.getByName(host)); System.out.
println("Server started."); Socket client = new Socket(host, port);
 System.out.println("Connecting to server..."); Socket connection =
 server.accept(); System.out.println("Connection established.");
DataOutputStream clientOut = new DataOutputStream(client.
getOutputStream()); DataInputStream clientIn = new DataInputStream(
client.getInputStream()); DataOutputStream serverOut = new
DataOutputStream(connection.getOutputStream()); DataInputStream
serverIn = new DataInputStream(connection.getInputStream()); System
.out.println("Communication is ready."); byte[] messageOut =
"Hello World".getBytes(); } }
```

2. **Object streams**

```
import java.io.ObjectInputStream; import java.io.ObjectOutputStream
; import java.net.InetAddress; import java.net.ServerSocket; import
 java.net.Socket; public class NetworkAppExample { public static
void main(String[] args) throws Exception { String host =
"localhost"; int port = 10430; ServerSocket server = new
ServerSocket(port, 50, InetAddress.getByName(host)); System.out.
println("Server started."); Socket client = new Socket(host, port);
 System.out.println("Connecting to server..."); Socket connection =
 server.accept(); System.out.println("Connection established.");
ObjectOutputStream clientOut = new ObjectOutputStream(client.
getOutputStream()); ObjectOutputStream serverOut = new
ObjectOutputStream(connection.getOutputStream()); ObjectInputStream
 clientIn = new ObjectInputStream(client.getInputStream());
ObjectInputStream serverIn = new ObjectInputStream(connection.
getInputStream()); System.out.println("Communication is ready.");
String messageOut = "Hello World"; } }
```

13. 

Send the message. Write data to the output stream and flush the stream to ensure the data has been written entirely.

1. **Data streams**
   Length of a message needs to be sent first so the other party knows how many bytes it needs to read. After the length is sent as a primitive integer type, bytes can be sent.

```
import java.io.DataInputStream; import java.io.DataOutputStream;
import java.net.InetAddress; import java.net.ServerSocket; import
java.net.Socket; public class NetworkAppExample { public static
void main(String[] args) throws Exception { String host =
"localhost"; int port = 10430; ServerSocket server = new
ServerSocket(port, 50, InetAddress.getByName(host)); System.out.
println("Server started."); Socket client = new Socket(host, port);
 System.out.println("Connecting to server..."); Socket connection =
 server.accept(); System.out.println("Connection established.");
DataOutputStream clientOut = new DataOutputStream(client.
getOutputStream()); DataInputStream clientIn = new DataInputStream(
client.getInputStream()); DataOutputStream serverOut = new
DataOutputStream(connection.getOutputStream()); DataInputStream
serverIn = new DataInputStream(connection.getInputStream()); System
.out.println("Communication is ready."); byte[] messageOut =
"Hello World".getBytes(); clientOut.writeInt(messageOut.length);
clientOut.write(messageOut); clientOut.flush(); } }
```

2. **Object streams**

```
import java.io.ObjectInputStream; import java.io.ObjectOutputStream
; import java.net.InetAddress; import java.net.ServerSocket; import
 java.net.Socket; public class NetworkAppExample { public static
void main(String[] args) throws Exception { String host =
"localhost"; int port = 10430; ServerSocket server = new
ServerSocket(port, 50, InetAddress.getByName(host)); System.out.
println("Server started."); Socket client = new Socket(host, port);
 System.out.println("Connecting to server..."); Socket connection =
 server.accept(); System.out.println("Connection established.");
ObjectOutputStream clientOut = new ObjectOutputStream(client.
getOutputStream()); ObjectOutputStream serverOut = new
ObjectOutputStream(connection.getOutputStream()); ObjectInputStream
 clientIn = new ObjectInputStream(client.getInputStream());
ObjectInputStream serverIn = new ObjectInputStream(connection.
getInputStream()); System.out.println("Communication is ready.");
String messageOut = "Hello World"; clientOut.writeObject(messageOut
); clientOut.flush(); } }
```

14. 

Log sent message. For logging purposes, print to the console that message has been sent.

1. **Data streams**

```
import java.io.DataInputStream; import java.io.DataOutputStream;
import java.net.InetAddress; import java.net.ServerSocket; import
```

```
java.net.Socket; public class NetworkAppExample { public static
void main(String[] args) throws Exception { String host =
"localhost"; int port = 10430; ServerSocket server = new
ServerSocket(port, 50, InetAddress.getByName(host)); System.out.
println("Server started."); Socket client = new Socket(host, port);
 System.out.println("Connecting to server..."); Socket connection =
 server.accept(); System.out.println("Connection established.");
DataOutputStream clientOut = new DataOutputStream(client.
getOutputStream()); DataInputStream clientIn = new DataInputStream(
client.getInputStream()); DataOutputStream serverOut = new
DataOutputStream(connection.getOutputStream()); DataInputStream
serverIn = new DataInputStream(connection.getInputStream()); System
.out.println("Communication is ready."); byte[] messageOut =
"Hello World".getBytes(); clientOut.writeInt(messageOut.length);
clientOut.write(messageOut); clientOut.flush(); System.out.println(
"Message sent to server: " + new String(messageOut)); } }
```

2. **Object streams**

```
import java.io.ObjectInputStream; import java.io.ObjectOutputStream
; import java.net.InetAddress; import java.net.ServerSocket; import
 java.net.Socket; public class NetworkAppExample { public static
void main(String[] args) throws Exception { String host =
"localhost"; int port = 10430; ServerSocket server = new
ServerSocket(port, 50, InetAddress.getByName(host)); System.out.
println("Server started."); Socket client = new Socket(host, port);
 System.out.println("Connecting to server..."); Socket connection =
 server.accept(); System.out.println("Connection established.");
ObjectOutputStream clientOut = new ObjectOutputStream(client.
getOutputStream()); ObjectOutputStream serverOut
```