# EXPLANATION

* **All the classes related to sockets are in the java.net package, so I made**

**sure to import that package when you program sockets.**

* **All the input/output stream classes are in the java.io package, include**

**this also**

* **How to open a socket?**
  + As programming a client, i created an object of

Socket class

* + Machine name is the machine i am trying to open a connection to,
  + PortNumber is the port (a number) on which the server you are trying to

connect to is running. I selected one that is 1500.

* **As programming a server, then this is how i open a socket:**
* ServerSocket MyService;
* try {
* MyServerice = new ServerSocket(PortNumber);
* }
* catch (IOException e) {
* System.out.println(e);
* }

**When implementing a server I also need to create a socket object from the ServerSocket in order to listen for and accept connections from clients.**

Socket clientSocket = null;

try {

clientSocket = MyService.accept();

}

catch (IOException e) {

System.out.println(e);

}

* **How to create an input stream?**
  + On the client side, i used the DataInputStream class to create an

input stream to receive response from the server:

* + The class DataInputStream allows you to read lines of text and Java

primitive data types in a portable way. It has methods such as read,

readChar, readInt, readDouble, and readLine,.

* + On the server side, you can use DataInputStream to receive input from

the client:

* **How to create an output stream?**
  + On the client side, I created an output stream to send information

to the server socket using the class PrintStream or DataOutputStream

of java.io:

* + The class PrintStream has methods for displaying textual representation

of Java primitive data types. Its write and println methods are important.

Also, you may want to use the DataOutputStream:

DataOutputStream output;

try {

output = new DataOutputStream(MyClient.getOutputStream());

}

catch (IOException e) {

System.out.println(e);

}

Many of its methods write a single Java primitive type to the output stream.

The method writeBytes is a useful one.

* **How to close sockets?**
  + You should always close the output and input stream before you close

the socket.

* + On the client side:
* try {
* output.close();
* input.close();
* MyClient.close();
* }
* catch (IOException e) {
* System.out.println(e);
* }
  + On the server side:
* try {
* output.close();
* input.close();
* clientSocket.close();
* MyService.close();
* }
* catch (IOException e) {
* System.out.println(e);
* }