Java Networking

Objectives

At the end of this module you should be able to

- Describe ports and sockets
- Describe clients and servers
- Write a `ServerSocket` class
- Write a client `Socket` class
- Read from a URL

Networking And I/O

Networking in Java is a natural extension to I/O.

- Making the network connection is simple
- Network connections provide access to streams
- Network connections are represented primarily as
  - java.net.Socket
  - java.net.ServerSocket
  - java.net.URL
- Many constructors and methods throw exceptions such as
  - java.net.MalformedURLException
  - java.net.UnknownHostException
  - java.net.BindException
- Most exceptions in java.net are sub-classes of java.io.IOException

Clients & Server

- **Port**
  - Each server is assigned a unique port number to use where it listens for connection requests
  - By convention, the use of ports 1 through 1024 is restricted to the operating system and standardized services
- **Sockets**
  - A software object that is created to represent a connection between two machines
  - Anytime a client and server connect a socket object is created on each machine

Identifying Hosts

```
import java.net.*;
   public class IPExample {
     public static void main(String[] args) {
       if(args.length != 1) {
          System.err.println("Usage: java IPExample MachineName");
          System.exit(0);
        }
        try {
           InetAddress ipAddress = InetAddress.getByName(args[0]);
           System.out.println(ipAddress);
         } catch (UnknownHostException e) {
           System.out.println("No IP address found for " + args[0]);
         }
       }
    }
```

Connecting to a Time Service

```
import java.io.*;
import java.net.*;
public class TimeServiceExample {
public static void main(String[] args) {
  try {
    Socket s = new Socket("nist1-ny.ustiming.org", 13);
    InputStream istrm = s.getInputStream();
    BufferedReader input = new BufferedReader(
                              new InputStreamReader(istrm));
    String line = null;
    do {
      line = input.readLine();
      if (line == null) {
        break;
      }
      System.out.println(line);
    } while (line != null);
    s.close();
  } catch (Exception e) {
    System.err.println(e.getMessage());}
  }
}
```

Writing a Server

1. A `ServerSocket` object is instantiated and listens at a specific port.
2. A client requests a connection.
3. If the `ServerSocket` accepts the connection, then its `accept()` returns a `Socket` that will be the server end of the connection.
4. The connection is established.
5. Objects of type `InputStream` and `OutputStream` are acquired from the `Socket` object over which the network data transfers will take place .

Server Example

```
import java.net.*;
import java.io.*;
public class ServerSocketExample {
public static void main(String[] args) throws IOException {
  ServerSocket server = new ServerSocket(8099);
  System.out.println("Server started: " + server);
  try {
    // accept() tells the server to listen.
    // Program blocks until a client asks for a connection
    Socket connection = server.accept();
    //  Now we have a connection and we can continue.
    try {
      System.out.println( "Connection established: "+ connection);
      //  Create the input and output streams
      BufferedReader input = new BufferedReader(
                              new InputStreamReader(
                                connection.getInputStream()));
      PrintWriter output = new PrintWriter(
                             new BufferedWriter(
                              new OutputStreamWriter(
                                connection.getOutputStream())),true);
      // We now loop until the client quits the connection
      . . .
```

Server Example (cont.)

```
    while(true) {
          String s = input.readLine();
          if (s.equals("quit")) {
            break;
          }
          System.out.println("Client said: " + s);
          output.println("You said "+ s);
        }
      //  Make sure the system resources are released
      } finally {
        System.out.println("Closing connection...");
        connection.close();
      }
   } finally {
     System.out.println("Server shutdown...");
     server.close();
   }
  } //end main
} //end class
```

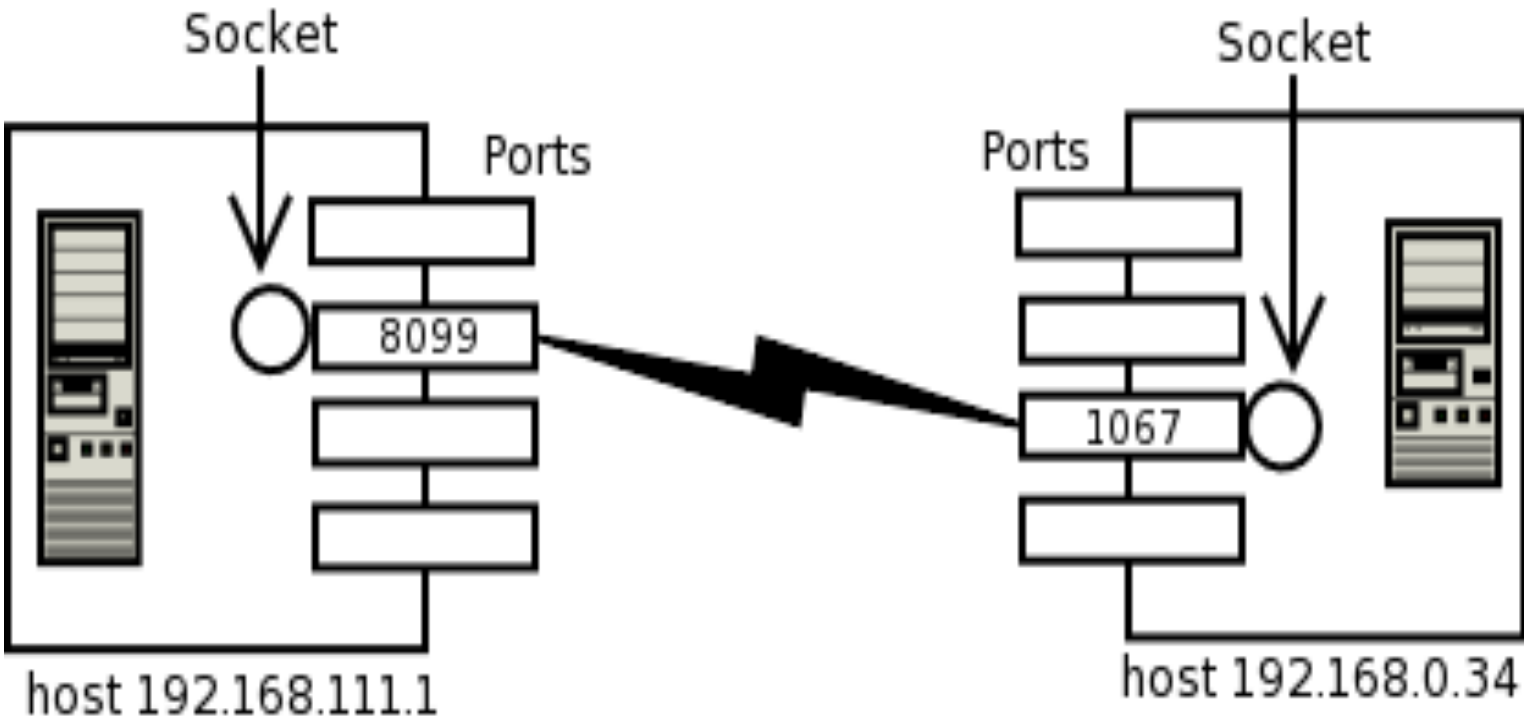
Client Example

```
import java.net.*;
import java.io.*;
public class ClientSocketExample {
  public static void main(String[] args) throws IOException {
    InetAddress addr = InetAddress.getByName(null);
    Socket connection = new Socket(addr, 8099);
    // Make sure we clean up the sockets now that we have a
    // socket connection
    try {
      System.out.println("connection socket = " + connection);
      BufferedReader input =new BufferedReader(
      new InputStreamReader(connection.getInputStream()));
      PrintWriter output = new PrintWriter(
                           new BufferedWriter(
                            new OutputStreamWriter(
                             connection.getOutputStream())),true);
```

Client Example (cont.)

```
      for (int i = 0; i < 10; i++) {
        output.println("Client generated line " + i);
        String s = input.readLine();
        System.out.println(s);
      }
      // Now we quit the connection
      output.println("quit");
    } finally {
      //  return system resources
      System.out.println("Closing connection...");
      connection.close();
    }
  } //end main
} //end class
```

Establishing Connection



An established connection with sockets on either end

Reading from a URL

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

public class URLReaderExample {
  public static void main(String[] args) throws Exception {
     // Open the connection and get a Reader
     URL url = new URL("http://www.google.com");
     BufferedReader in = new BufferedReader(
        new InputStreamReader(url.openStream()));
     // read from the URL
     String inputLine;
     while ((inputLine = in.readLine()) != null) {
       System.out.println(inputLine);
     }
     in.close();
  }
}
```

Summary

We covered

- Ports and sockets
- Clients and servers
- Writing a `ServerSocket` class
- Writing a client `Socket` class
- Reading from a URL