Module 13

Networking

Huang Hua

School of Computer & Information Technology
Beijing Jiaotong University
hhua@bjtu.edu.cn

Objectives

- URL programming
- Develop code to set up the network connection
- Understand the TCP/IP Protocol
- Use ServerSocket and Socket classes for implementation of TCP/IP clients and servers
- Data diagrams

Relevance

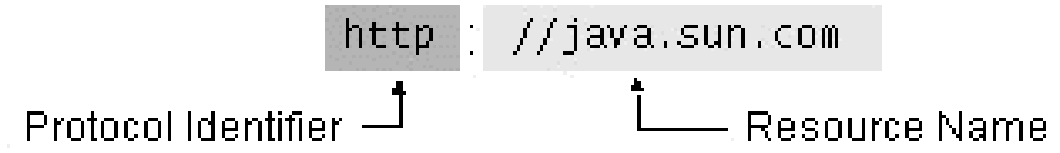
How can a communication link between a client machine and a server on the network be established?

Working with URLs

A URL takes the form of a string that describes how to find a resource on the Internet.

URLs have two main components:

- the protocol needed to access the resource
- and the location of the resource.



Host Name	The name of the machine on which the resource lives.
Filename	The pathname to the file on the machine.
Port Number	The port number to which to connect (typically optional).
Reference	A reference to a named anchor within a resource that usually identifies a specific location within a file (typically optional).

Working with URLs(Cont'd)

- Creating a URL

```
URL gamelan = new URL("http://www.gamelan.com/");
```

- Creating a URL Relative to Another

```
URL gamelan = new URL("http://www.gamelan.com/pages/");
URL gamelanGames = new URL(gamelan, "Gamelan.game.html");
URL gamelanNetwork = new URL(gamelan, "Gamelan.net.html");
```

- Other URL Constructors

```
new URL("http", "www.gamelan.com", /pages/Gamelan.net.html");
URL gamelan = new URL("http", "www.gamelan.com", 80,
                                   "pages/Gamelan.network.html");
```

- Parsing a URL

```
getProtocol                  getAuthority
getHost                      getPort
getPath                      getQuery
getFile                      getRef
```

URL Reading

- After creating a URL, the **openStream**() method of URL can be called to get a stream to read the contents of the URL.

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

public class URLReader {
  public static void main(String[] args) throws Exception {
  URL yahoo = new URL("http://www.yahoo.com/");
  BufferedReader in = new BufferedReader(
        new InputStreamReader(yahoo.openStream()));

    String inputLine;

    while ((inputLine = in.readLine()) != null)
      System.out.println(inputLine);

    in.close();
    }
  }
```

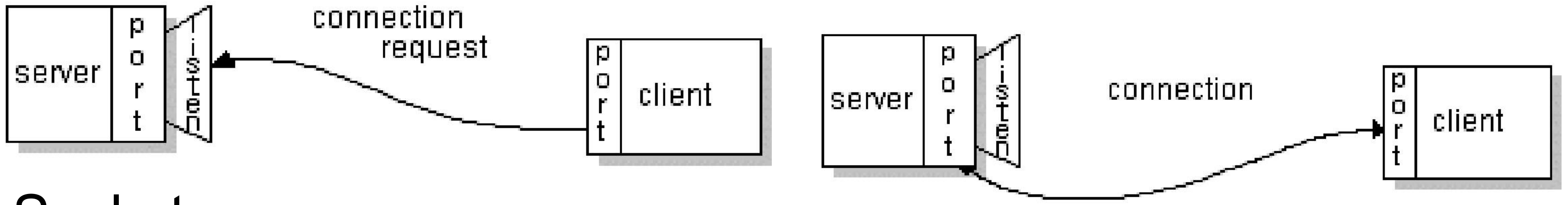
Reading from and Writing to a URLConnection

- After creating a URL object, **openConnection** method can be called to get a **URLConnection** object, or one of its protocol specific subclasses (eg, **HttpURLConnection**).
- **URLConnection** can be used for reading/writing.
  See demos for more info.

```
import java.net.*;
import java.io.*;
public class URLConnectionReader {
    public static void main(String[] args) throws Exception {
        URL yahoo = new URL("http://www.yahoo.com/");
        URLConnection yc = yahoo.openConnection();
        BufferedReader in = new BufferedReader(
           new InputStreamReader(yc.getInputStream()));
        String inputLine;
        while ((inputLine = in.readLine()) != null)
            System.out.println(inputLine);
        in.close();
    }
}
```

Networking

This section describes networking concepts.



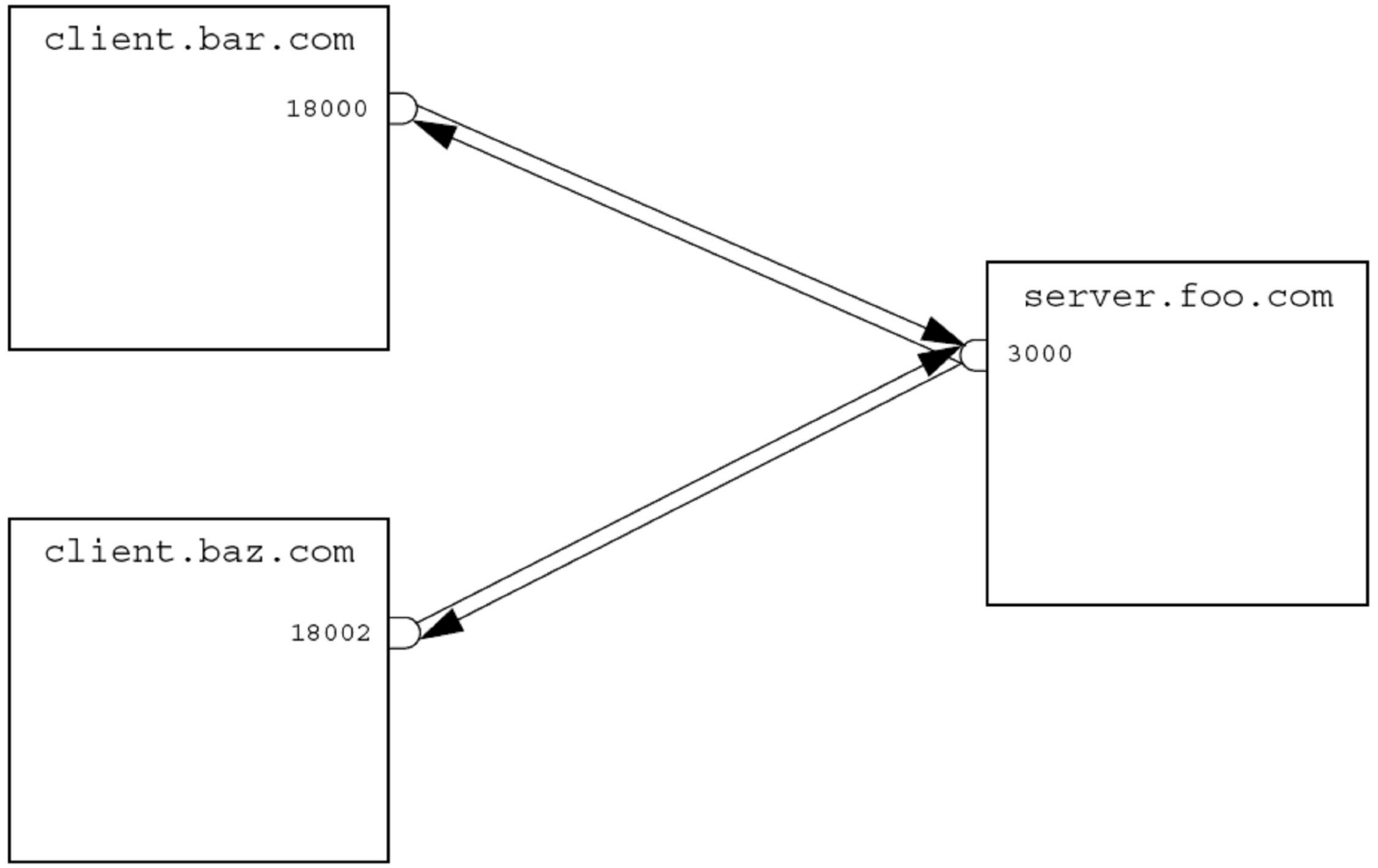
Sockets

- A socket is one endpoint of a two-way communication link between two programs running on the network.
- Sockets hold two streams: an **input stream** and an **output stream**.
- Each end of the socket has a pair of streams.

Setting Up the Connection

- Set up of a network connection is similar to a telephone system: One end must *dial* the other end, which must be *listening*.

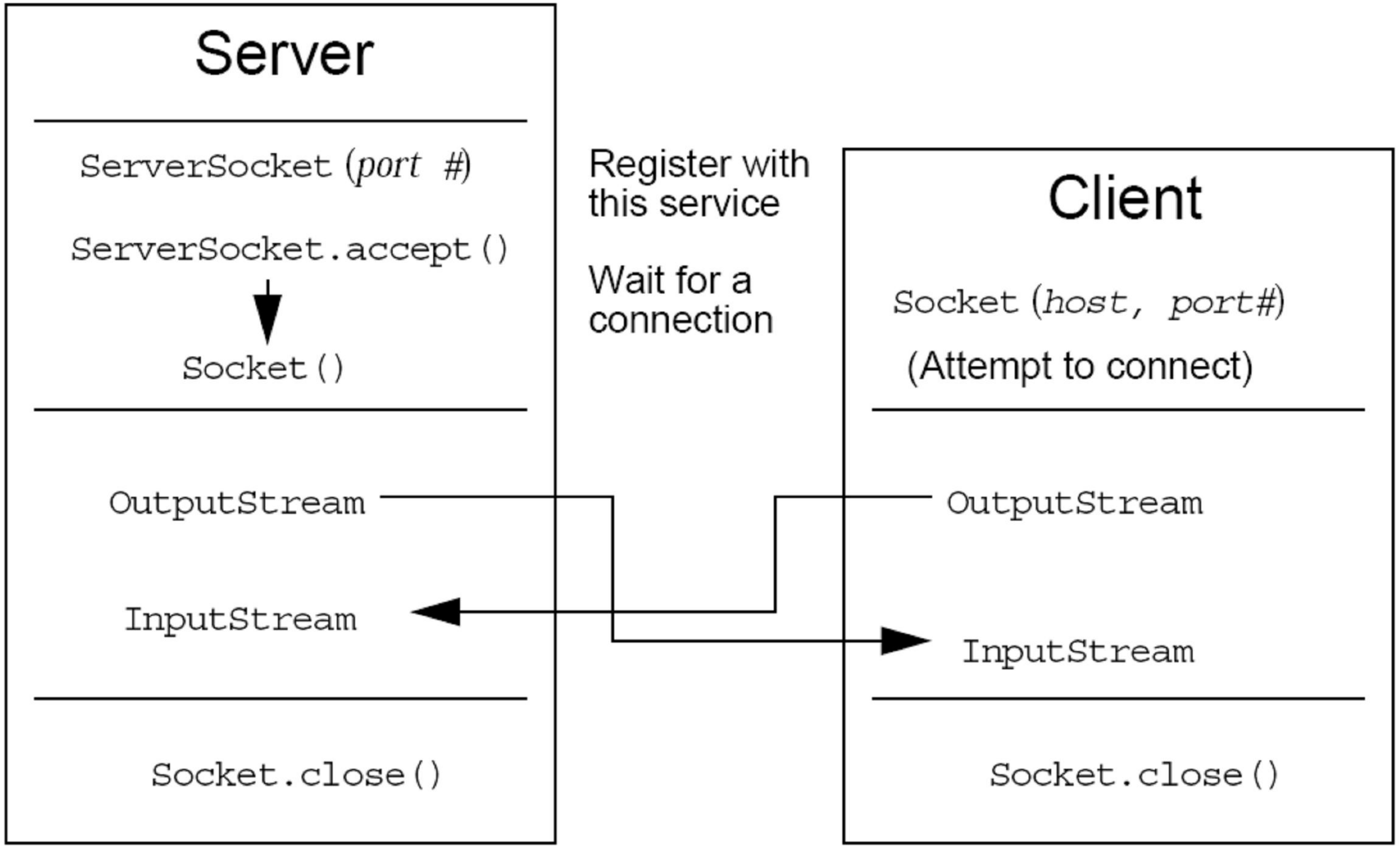
Networking(Cont’d)



Networking With Java Technology

- To address the connection, include the following:
  - The **address** or **name** of remote machine
  - A **port number** to identify the purpose at the server
- Port numbers range from 0–65535.
- A connection is identified by four numbers: client IP address, client port number, server IP address, and server port number.

Java Networking Model



Minimal TCP/IP Server

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

public class SimpleServer {
  public static void main(String args[]) {
    ServerSocket s = null;

    // Register your service on port 5432
    try {
      s = new ServerSocket(5432);
    } catch (IOException e) {
      e.printStackTrace();
    }

    // Run the listen/accept loop forever
    while (true) {
      try {
        // Wait here and listen for a connection
        Socket s1 = s.accept();
```

Minimal TCP/IP Server

```

        // Get output stream associated with the socket
        OutputStream s1out = s1.getOutputStream();
        BufferedWriter bw = new BufferedWriter(
          new OutputStreamWriter(s1out));

        // Send your string!
        bw.write("Hello Net World!\n");

        // Close the connection, but not the server socket
        bw.close();
        s1.close();

      } catch (IOException e) {
        e.printStackTrace();
      } // END of try-catch
    } // END of while(true)
  } // END of main method
} // END of SimpleServer program
```

Minimal TCP/IP Client

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

public class SimpleClient {

  public static void main(String args[]) {

    try {
      // Open your connection to a server, at port 5432
      // localhost used here
      Socket s1 = new Socket("127.0.0.1", 5432);

      // Get an input stream from the socket
      InputStream is = s1.getInputStream();
      // Decorate it with a "data" input stream
      DataInputStream dis = new DataInputStream(is);
```

Minimal TCP/IP Client

```

        // Read the input and print it to the screen
        System.out.println(dis.readUTF());

        // When done, just close the steam and connection
        dis.close();
        s1.close();

      } catch (ConnectException connExc) {
        System.err.println("Could not connect.");

      } catch (IOException e) {
        // ignore
      } // END of try-catch

    } // END of main method

  } // END of SimpleClient program
```

About Datagrams

- A **datagram** is an independent, self-contained message sent over the network whose arrival, arrival time, and content are not guaranteed.
- The **`java.net`** package contains three classes to help writing Java programs that use datagrams to send and receive packets over the network:
  - **`DatagramSocket`**, **`DatagramPacket`**, **`MulticastSocket`**
  - Application can send and receive **`DatagramPackets`** through a **`DatagramSocket`**.
  - **`DatagramPackets`** can be broadcast to multiple recipients all listening to a **`MulticastSocket`**.

Summary

- URL programming
- TCP programming using **`ServerSocket`** and **`Socket`** classes to implement TCP/IP **clients** and **servers**
- Data diagrams programming

For Further Study

- Database & JDBC DB Access
- JavaFX & 2/3D graphics
- Multimedia and media processing
- Package with Jar & Web Start
- Class & Reflection
- Security features
- Networking

For Further Reading:

- Effective Java(2ed)，大话设计模式
- 挑战编程，算法竞赛入门经典（训练指南），编程珠玑，编程之美
- 数学之美

Questions or Comments?