

Introduction to Java Network Programming

by Derek Schuurman
Redeemer University College

1

Transport Layer Services

- The transport layer also provides an interface for application programmers
 - Hides underlying details of the network layer
- The transport layer on the Internet uses two protocols (we will discuss these later):
 1. TCP
 2. UDP

2

Sockets

- The Berkeley sockets interface was originally developed at the University of California at Berkeley as a tool to for network programming
- A **Socket** is a handle to a communications link over a network with another application
 - A socket connection includes Local IP address and Port number and a Remote IP address and Port number



3

Java Networking

- Java is one of the first languages designed with networking in mind
- Network programming in Java is easy!
 - Java applications can easily send and receive data across the Internet
 - The **java.net** package provides the classes for implementing networking applications
 - can be included with `import java.net.*;`
 - see the **java.net** summary in the Java API

4

Some **java.net** Classes

- **ServerSocket** - implements server sockets
- **Socket** - implements client sockets
- **InetAddress** - represents an IP Address
- **DatagramPacket** - represents a UDP datagram packet
- **DatagramSocket** - socket for sending and receiving UDP datagram packets
- **MulticastSocket** - used for sending and receiving IP multicast packets
- **URL** - represents a Uniform Resource Locator, a pointer to a "resource" on the World Wide Web

5

Java ServerSocket Class

- **ServerSocket Class**
 - used by servers to listen for clients
 - Constructor: **ServerSocket(int port)**
 - Creates a server socket on the specified port
 - methods include:
 - **accept**
 - Listens for a connection to be made and accepts it
 - This method *blocks* until a connection is made
 - Returns a socket for communication with the client
 - **close**
 - Closes the socket
- See the ServerSocket description in the Java API

6

Making a Client Connection

- A Client needs to perform 5 steps:
 1. Open a socket
 2. Open an input stream and output stream to the socket
 3. Read from and write to the stream according to the server's protocol
 4. Close the streams
 5. Close the socket

7

Java Socket Class

- Socket Class
 - Used by clients to talk to servers
 - Used by servers to talk to clients
 - methods include:
 - **close**
 - When you are done communicating you should close the socket
 - **connect**
 - Connects this socket to a server (connections can also be established using the constructor method)
 - A connection can also be established using constructor parameters that specify the host and port
 - See Socket class description in the Java API

8

Reading and Writing Socket Data

- The socket class has methods to obtain the input and output streams
 - **getInputStream()**
 - Returns the **InputStream** for the socket
 - an input stream of bytes
 - **getOutputStream()**
 - Returns the **OutputStream** for the socket
 - an output stream of bytes

9

Reading and Writing Socket Data

- **InputStream** and **OutputStream** deals with *bytes* which is not very convenient
- To make application coding easier, streams can be handled by other objects:
 - The input stream can be handled using the **Scanner** class
 - Provides various method for reading strings etc.
 - The output streams can be handled using the **PrintWriter** class
 - Provides a **println** method for output of strings

10

Server Socket Example in Java

```
// Create a new server socket on port 7777
ServerSocket server = new ServerSocket(7777);

// wait for a client connection
Socket s = server.accept();

// Create an object for reading data from the client
Scanner in = new Scanner(s.getInputStream());

// Create an object for writing to the client
PrintWriter out = new PrintWriter(s.getOutputStream());

// Get data from the client
String msg = in.nextLine();

// Send some data to the client
out.println("Hello from the CS server");
out.flush(); // empties the buffer
```

11

Client Socket Example in Java

```
// Create a socket connection to CS server port 7777
Socket s = new Socket("cs.redeemer.ca", 7777);

// Create an object for reading data from the server
Scanner in = new Scanner(s.getInputStream());

// Create an object for writing to the server
PrintWriter out = new PrintWriter(s.getOutputStream());

// Send some data to the CS server
out.println("Hello CS server");
out.flush();

// Get reply from CS server
String reply = in.nextLine();
```

12

Sending UDP Packets in Java

- The following classes are used to send UDP packets in Java
 - **DatagramSocket**
 - class provides a socket for sending and receiving datagram packets
 - **DatagramPacket**
 - represents a datagram packet
 - **MulticastSocket**
 - Inherited from the DatagramSocket class
 - Includes capabilities for sending and receiving multicast packets

13

UDP Example

```
// create a datagram socket
DatagramSocket socket = new DatagramSocket();

// wait for a packet
byte[] buf = new byte[256];
packet = new DatagramPacket(buf, buf.length);
socket.receive(packet);

// display response
String received = new String(packet.getData());
System.out.println("Reply=" + received);
```

14

Network Data

- Besides bytes and strings, other types of data can be sent through a socket including:
 - Objects
 - Requires object serialization
 - Images
 - Multimedia
 - Compressed data
 - Encrypted data

15

Some Methods of InetAddress Class

- The **InetAddress** class represents an IP address
- This class includes methods to:
 - Determine your IP address
 - Resolve IP addresses from hostnames
 - Find hostnames given an IP addresses

16

What is my IP Address?

- There are several ways to determine the IP address for a computer:
 - in a Linux shell, type:
 - **/sbin/ifconfig** (Linux)
 - **ipconfig** (Windows)
 - Using the **InetAddress** Class in Java:

```
InetAddress local = InetAddress.getLocalHost();
System.out.println("IP Address:"+local.getHostAddress());
```

17

Finding IP Addresses from Hostnames

- The **InetAddress** class can be used to resolve an IP address given a hostname

```
InetAddress inet =
    InetAddress.getByName("cs.redeemer.ca");
System.out.println ("IP Address:" +
    inet.getHostAddress());
```

- the **getHostAddress()** method returns the IP address in a string in decimal dot notation

18

Finding Hostname from IP Addresses

- The **InetAddress** class can also be used to find the hostname given an IP address

```
InetAddress inet =  
InetAddress.getByName("205.211.11.1");  
System.out.println("Host: " +  
inet.getHostName());
```

- the `getHostName()` method returns the hostname

19

Connection Problems

- Various problems can arise when attempting to connect to a server
 - Wrong IP address or hostname cannot be resolved
 - Wrong port numbers
 - Internal vs. external addresses and NAT
 - ↳ e.g. internally, the alpha server is `10.0.0.1`, the external address is `205.211.11.1`
 - Firewalls
 - ↳ helpful for security, annoying for developers
 - Congestion etc.
- Connection problems can be handled using java *exceptions*

20

Exceptions

- Exceptions are surprising events that occur during program execution
 - For example:
 - ↳ **ConnectException** - an error occurred while attempting to connect to a remote port
- In Java, Exceptions are events that are thrown by code that encounters an unexpected condition
- Exceptions are then caught by code that "handles" the exception
- the **java.net** package includes several exceptions to handle networking errors

21

java.net Exceptions

- Other exceptions include:
 - **BindException** - an error occurred while attempting to bind a socket to a local port
 - **NoRouteToHostException** - connection error
 - **PortUnreachableException** - an ICMP Port Unreachable message received
 - **ProtocolException** - an error in the underlying protocol
 - **UnknownHostException** - the IP address of a host could not be determined

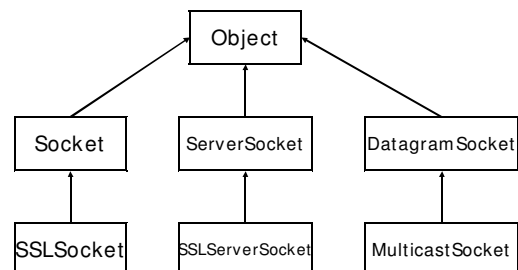
22

URLs

- An **URL** is an acronym for *Uniform Resource Locator*
 - An URL is a reference to a resource on the World Wide Web
 - ↳ Defined by **RFC 2396**
 - The **URL** class can be used to reference URLs on the web
 - If contains information such as
 - ↳ Hostname
 - ↳ File name
 - ↳ Port
 - ↳ protocol

23

Partial Class Hierarchy



24