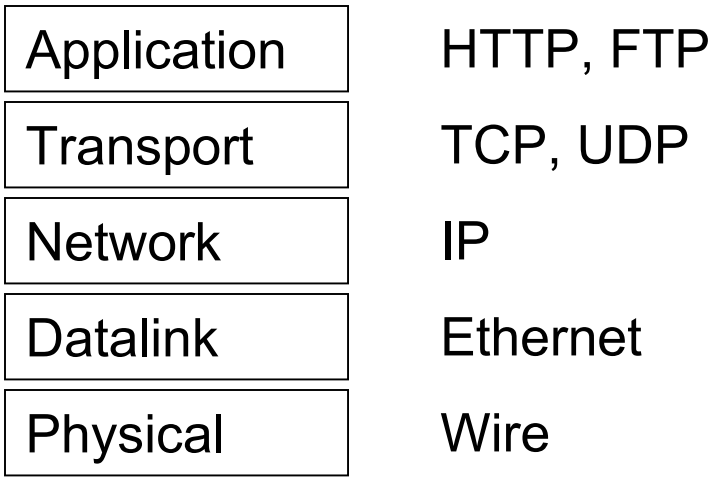
Simple Network Programming

Objectives

- Learn how to program simple network clients and servers.
- Learn how to use the Command pattern.

Basics of TCP/IP

- Five layer stack that represents the network:



Java allows you to address the Application and Transport layers

Basics of TCP/IP (cont'd)

- Application Layer
  - HTTP protocol for transmitting hypertext
  - FTP protocol for file transfer
  - Telnet for remote login
  - NNTP for news, SMTP for mail
- Each protocol has different text "commands" that both server and client understand

Basics of TCP/IP (cont'd)

- Transport Layer
  - TCP: reliable, connection-based protocol for transmitting packets
  - UDP: unreliable, connectionless protocol for transmitting packets
- TCP has lots more overhead than UDP

Basics of TCP/IP (cont'd)

- Network Layer
  - IP protocol defines address space
  - Abstracts the underlying hardware

Basics of TCP/IP (cont'd)

- Datalink Layer
  - Ethernet
  - Converts data understood by machine into voltage differential, appropriate for sending over the wire
  - Similar to the modem concept (converts digital data to analog signals)

Basics of TCP/IP (cont'd)

- Addresses and Ports
  - Each device has a 32-bit address (IPv4)
  - Addresses can also be registered with a name
  - DNS/Bind translates names into addresses
  - Each device has a 15-bit port space (32,768 ports)
  - Port is not a physical notion--it's a software abstraction

Basics of TCP/IP (cont'd)

- TCP/IP under Windows
  - Ethernet frames are created at the ISP on the other side of their modem
  - If you have a dialup to an ISP, you may have to connect to do network programming
  - You can tell what your current (dynamically-assigned) IP address is by running ipconfig

Client-server

- Two computers communicating
- Allows for resource reuse
- Allows for load-balancing--buy one super powerful computer instead of 1000 less powerful ones
- Allows centralized resource managment
- Server is a central repository of information
- Client interacts with user, and is as “thin” as possible

Client-server (cont'd)

- Java networking is based on client-server
- Server
  - sometimes already exists (Apache, Tomcat, sendmail)
  - sometimes you have to create it for your application
- Client
  - you usually are creating this

Client-server (cont'd)

- A typical TCP/IP session:
  - Client connects to well-known port on server
    - The server can support multiple connections on a single port
  - Client and server communicate over this connection via streams
  - When communication is complete the connection and streams are closed

Package java.net

- Several important classes:
  - InetAddress: models an IP address
  - URL: models a URL
  - Socket: models a low-level connection to a port on another machine
  - ServerSocket: listens for socket connect requests on a port, then creates a private connection for the communication

java.net.InetAddress

- Gets an IP address by name or number
- Allows you to do a DNS lookup
- getHostAddress() gets the numeric address
- getHostName() gets the host name
- static getLocalHost() returns this nodes address or the special "loopback" address 127.0.0.1

java.net.InetAddress

Example

```
public class WhoAmI {
    public static void main(String[] args) throws Exception {
        if(args.length != 1) {
            System.err.println("Usage: WhoAmI MachineName");
            System.exit(1);
        }
        InetAddress address = InetAddress.getByName(args[0]);
        System.out.println(address.toString());
        System.out.println(address.getHostAddress());
        System.out.println(address.getHostName());
    }
}
```

java.net.URL

- Represents a Uniform Resource Locator (URL)
- Example:
  - http://www.javasoft.com:80/index.html
- protocol://machine:port/file
- Each protocol has a default port
  - http is 80
  - ftp is 21
  - telnet is 23
  - smtp is 25

java.net.URL (cont'd)

- Constructor takes a String representing the URL (other forms are available)
- Useful methods available to analyze the URL
- openStream() returns an InputStream that you can use to read the data
- openConnection() returns a java.net.URLConnection object--this gives finer control over reading the data, etc.
- data are returned as text--not formatted!

Socket

- A socket represents a connection between client and server
- Like a two-way pipe: sink for the sender, source for the receiver
- Note that both parties must be able to send and receive
- Modeled by java.net.Socket

Socket (cont'd)

- Class java.net.Socket
  - Constructor takes the machine and port to connect to
  - Methods to retrieve information (address, port of both parties, etc.)
  - getInputStream() and getOutputStream()

Socket (cont'd)

- Note how when you use custom sockets, you have to know the protocol
- This protocol will either be something you've invented, or an established standard
- Internet protocols are established by IETF Request for Comments (RFCs)
- There are already existing abstractions (implementations) of most established protocols

Socket (cont'd)

- Always close a socket when you are finished
- The call to close() should be in a finally block
- Sockets use system resources that cannot be garbage-collected

ServerSocket

- To handle your own protocol, you will need a client and server
- A server is implemented using a ServerSocket
- Modeled by java.net.ServerSocket
- The accept() method blocks, listening for a connection, and returns a Socket when a connection is established

ServerSocket (cont'd)

- General form of ServerSocket usage

```
    ServerSocket server = new
    ServerSocket(12345);
    while (true) {
        Socket connection = server.accept();
        InputStream in = connecion.getInputStream();
        OutputStream out = connecion.getOutputStream();
        // read and write to the streams
        ...
        connection.close();
    }
}
```

ServerSocket (cont'd)

- Most servers are multithreaded, otherwise only one client could be handled at a time
- This is okay--you really should be encapsulating your server behavior in a separate class anyway
- The usual concerns of concurrent programming apply, of course
- Also, you might want to have an “admin” thread in a real-world server, so you don’t have to shutdown with Ctrl-C

UDP vs. TCP

- A socket connection is just that--a connection
- It is a dedicated link between client and server
- It will only close when the parties request it
- Data are guaranteed to arrive at the destination, and in the order sent
- These are all features supplied by TCP

UDP vs. TCP (cont'd)

- UDP, on the other hand, is connectionless
- UDP=User Datagram Protocol
- If TCP is like a phone conversation, UDP is like US Mail or carrier pigeon
- UDP packages data in discrete packets (limit of 64K) and sends them over the network
- They will not necessarily arrive in the order sent, and there are no guarantees they will arrive at all!

UDP vs. TCP (cont'd)

- So why use UDP?
- Sometimes you care about speed more than quality
- UDP has a lot less overhead than TCP
- RealAudio is UDP-based. You care more that the transmission keeps up than that every single bit of audio gets through

UDP vs. TCP (cont'd)

- UDP has no ServerSocket notion, because there is no connection to set up
- Class java.net.DatagramSocket represents a UDP socket
- method receive() waits for a DatagramPacket to show up
- method send() sends a DatagramPacket
- A DatagramPacket can be reused

DatagramPacket

- One constructor for sending:

```
DatagramPacket(byte[] buf, int len,
               InetAddress destaddr,
               int destport)
```

- One constructor for receiving:

```
DatagramPacket(byte[] buf, int len)
```

- Set/get methods for important attributes: data (buf), length, address, and port

Using the Command Design Pattern

- The Command design pattern: "encapsulates a request as an object, allowing parameterizing clients with different requests"
- The Command pattern is superior to using instanceof or some other protocol to determine what action the server is to perform

Command Problem

- Desirable for an object to make a request of another object
  - Without being concerned with what the other object is
  - Or what action it is going to perform
  - A menu system is a popular example of this

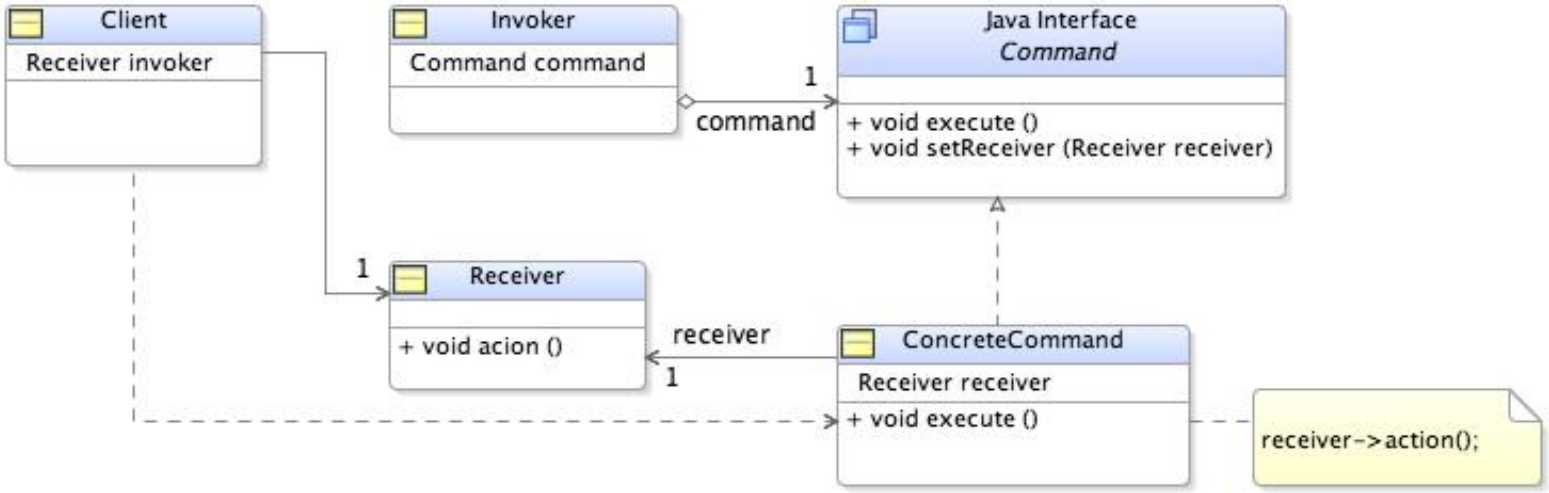
Command Solution

- Define a class to carry out a specific command
- Command object performs command on behalf of command receiver
- Receiver must be configured with command
- Command takes action on some other object
  - Must be configured with this object

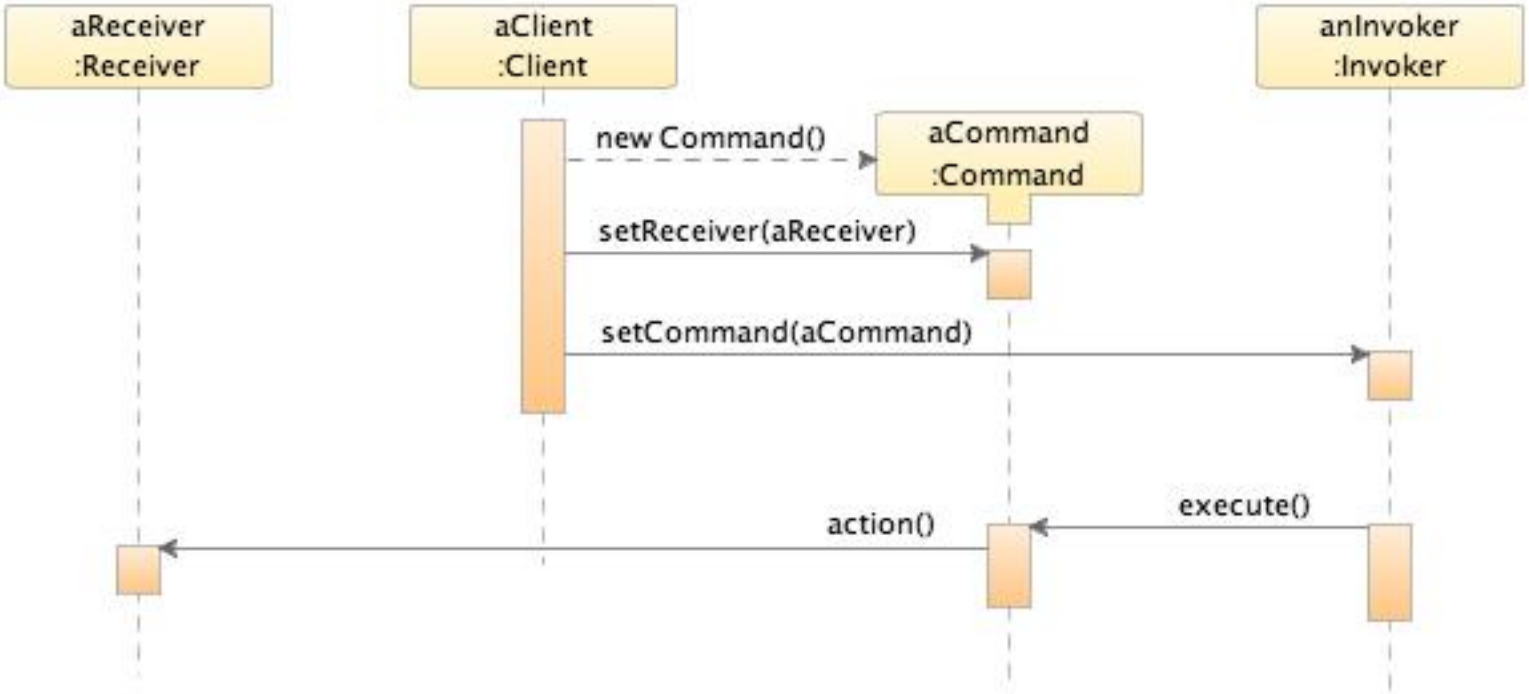
Classes in the Command Pattern

- Command – declares an interface for executing an operation
- ConcreteCommand -- Binds a Receiver object and an action request
- Client – creates a ConcreteCommand and sets its receiver
- Invoker – asks the command to carry out the request
- Receiver – knows how to perform the operations required to carry out the request

Command Structure



Command Interaction



Command Solution

- Defining characteristics
  - An interface (or abstract class) is defined for command objects
  - The Invoker is configured with objects realizing the Command interface

Command Solution

- Discussion
  - An example
    - Application wants to specify actions by name
    - Command pattern is well suited to this
    - Introduce a map to lookup commands