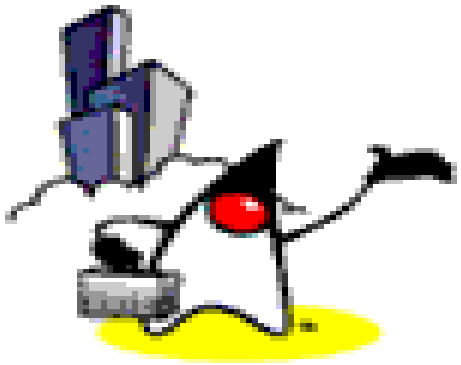
Java Networking API

Sang Shin
Founder and Chief Instructor
www.JPassion.com
"Learn with JPassion!"

Topics

- Basic Concepts on Networking
  - IP Address
  - Protocol
  - Ports
  - The Client/Server Paradigm
  - Sockets
- The Java Networking Package
  - The *ServerSocket* class and the *Socket* class
  - The *MulticastSocket* class and the *DatagramPacket* class



Basic Concepts of Networking

IP Address

- Conceptually similar to the traditional mailing address
  - Each house has a traditional mailing address
  - Each computer connected to the Internet has a unique IP address
- A 32-bit number (IPv4) or 128-bit number (Ipv6) used to uniquely identify each computer connected to the Internet
  - 192.1.1.1 (IP address)
  - docs.rinet.ru (Hostname)

Communication Protocol

- Why protocols?
  - Different types of communication occurring over the Internet
  - Each type of communication requires a specific and unique protocol
- Definition
  - Set of rules and standards that define a certain type of Internet communication
  - Describes the following information:
    - Format of data being sent over the Internet
    - How it is sent
    - When it is sent

Communication Protocol

- Not entirely new to us. Consider this type of conversation:

  "Hello."

  "Hello. Good afternoon. May I please speak at Joan?"

  "Okay, please wait for a while."

  "Thanks."

  ...

  - Social protocol used in a telephone conversation
  - Gives us confidence and familiarity of knowing what to do

Communication Protocol

- Some important protocols used over the Internet
  - Hypertext Transfer Protocol (HTTP)
    - Used to transfer HTML documents on the Web
  - File Transfer Protocol (FTP)
    - More general compared to HTTP
    - Allows you to transfer binary files over the Internet
  - Both protocols have their own set of rules and standards on how data is transferred
  - Java provides support for both protocols

Ports

- Protocols only make sense when used in the context of a service
  - HTTP protocol is used when you are providing Web content through an HTTP service
  - Each computer on the Internet can provide a variety of services

- Why Ports?
  - The type and address of service must be known before information can be transferred

Ports

- Definition:
  - A 16-bit number that identifies each service offered by a network server
- Using a particular service to establish a line of communication through a specific protocol
  - Need to connect to the appropriate port

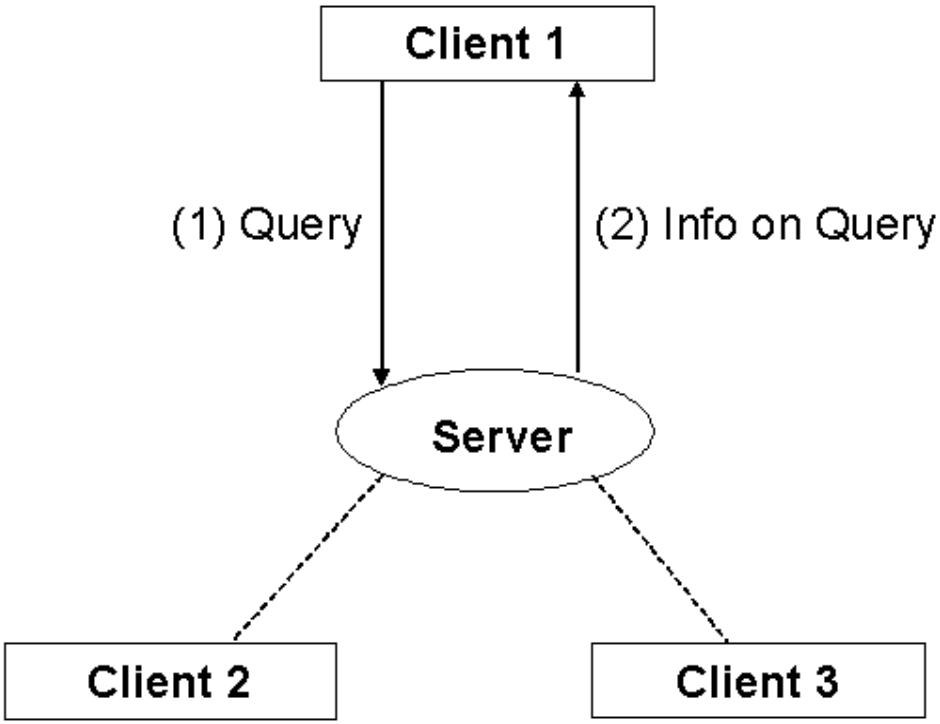
Ports

- Standard ports
  - Numbers specifically associated with a particular type of service
  - Examples:
    - The FTP service is located on port 21
    - The HTTP service is located on port 80 or 8080
  - Given port values below 1024
- Port values above 1024
  - Available for custom communication
  - If already in use by some custom communication, you must look for other unused values

The Client/Server Paradigm

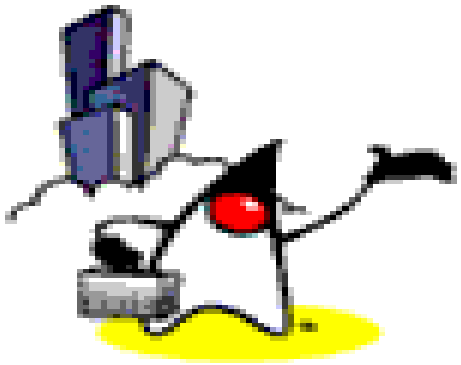
- Basis for Java networking framework
- Involves two major elements:
  - Client
    - Machine in need of some type of service
  - Server
    - Machine providing service and waiting for a request
- Scenario:
  - Client connects to a server and queries for certain information
  - Server considers the query and returns information on it to the client

The Client/Server Paradigm



What is a Socket?

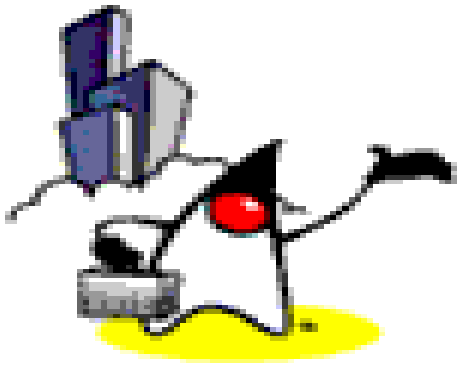
- Software abstraction for an input or output medium of communication
- Communication channels that enable you to transfer data through a particular port
- An endpoint for communication between two machines
- A particular type of network communication used in most Java network programming
- Java performs all of its low-level network communication through sockets



Java Networking Package

The Java Networking Package

- The *java.net* package
- Provides classes useful for developing networking applications
- Some classes in the package:
  - *ServerSocket*
  - *Socket*
  - *MulticastSocket*
  - *DatagramPacket*



ServerSocket Class

The *ServerSocket* Class

- A server socket waits for requests to come in over the network

ServerSocket Constructors
`ServerSocket(int port)`
Instantiates a server that is bound to the specified port. A port of 0 assigns the server to any free port. Maximum queue length for incoming connection is set to 50 by default.
`ServerSocket(int port, int backlog)`
Instantiates a server that is bound to the specified port. Maximum queue length for incoming connection is is based on the *backlog* parameter.

The *ServerSocket* Class: Methods

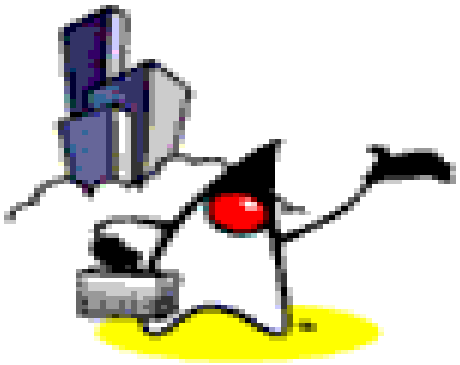
ServerSocket Methods
`public Socket accept()`
Causes the server to wait and listen for client connections, then accept them.
`public void close()`
Closes the server socket. Clients can no longer connect to the server unless it is opened again.
`public int getLocalPort()`
Returns the port on which the socket is bound to.
`public boolean isClosed()`
Indicates whether the socket is closed or not.

The *ServerSocket* Class: Example

```
1  import java.net.*;
2  import java.io.*;
3  public class EchoingServer {
4     public static void main(String [] args) {
5        ServerSocket serverSocket = null;
6        Socket socket;
7        try {
8           // Let's assume 1234 is available port
9           serverSocket = new ServerSocket(1234);
10       } catch (IOException ie) {
11          System.out.println("Cannot open socket.");
12          System.exit(1);
13       }
14 //continued...
```

The *ServerSocket* Class: Example

```
15          while(true) {
16              try {
17                  socket = serverSocket.accept();
18                  OutputStream serverOut =
19                              socket.getOutputStream();
20                  PrintWriter pw =
21                              new PrintWriter(serverOut, true);
22                  InputStream serverIn =
23                              socket.getInputStream();
24                  BufferedReader br = new BufferedReader(new
25                                      InputStreamReader(serverIn));
26                  pw.println(br.readLine());
27              } catch (IOException ie) {}}}
28  }
```



Socket Class

The *Socket* Class

- A socket is an endpoint for communication between two machines.

Socket Constructors
`Socket(String host, int port)`
Creates a client socket that connects to the given port number on the specified host.
`Socket(InetAddress address, int port)`
Creates a client socket that connects to the given port number at the specified IP address.

The *Socket* Class: Methods

Socket Methods
`public void close()`
Closes the client socket.
`public InputStream getInputStream()`
Retrieves the input stream associated with this socket.
`public OutputStream getOutputStream()`
Retrieves the output stream associated with this socket.
`public InetAddress getInetAddress()`
Returns the IP address to which this socket is connected
`public int getPort()`
Returns the remote port to which this socket is connected.
`public boolean isClosed()`
Indicates whether the socket is closed or not.

The *Socket* Class: Example

```
1  import java.io.*;
2  import java.net.*;
3
4  public class MyClient {
5     public static void main(String args[]) {
6        try {
7           /* Socket client = new Socket("133.0.0.1",
8                                                 1234); */
9           Socket client =
10                new Socket(InetAddress.getLocalHost(),
11                           1234);
12 //continued...
```

The *Socket* Class: Example

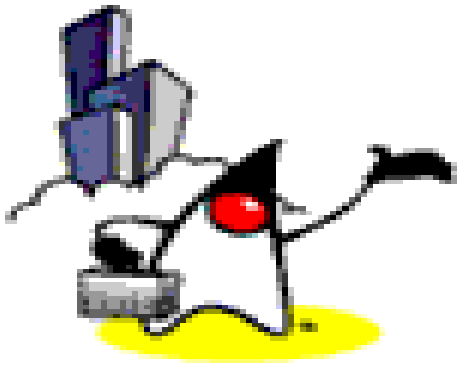
```
13              InputStream clientIn =
14                          client.getInputStream();
15              OutputStream clientOut =
16                          client.getOutputStream();
17              PrintWriter pw = new PrintWriter(clientOut,
18                                               true);
19              BufferedReader br = new BufferedReader(new
20                                  InputStreamReader(clientIn));
21              BufferedReader stdIn = new BufferedReader(new
22                                  InputStreamReader(System.in));
23              System.out.println("Type a message for
24                                  the server: ");
25  //continued...
```

The *Socket* Class: Example

```
26              pw.println(stdIn.readLine());
27              System.out.println("Server message: ");
28              System.out.println(br.readLine());
29              pw.close();
30              br.close();
31              client.close();
32          } catch (ConnectException ce) {
33              System.out.println("Cannot connect to
34                                  the server.");
35          } catch (IOException ie) {
36              System.out.println("I/O Error.");
37          }
38      }
39  }
```

Lab:

Exercise 1: Client nd Server
1026_javase_networking.zip



DatagramPacket Class

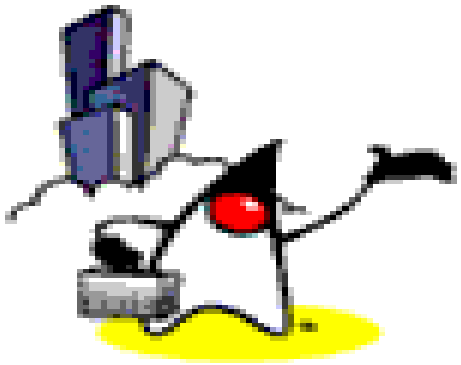
The *DatagramPacket* Class

- Used to deliver data through a connectionless protocol
- Delivery of packets is not guaranteed

DatagramPacket Constructors
`DatagramPacket(byte[] buf, int length)`
Constructs a datagram packet for receiving packets with a length *length*. *length* should be less than or equal to the size of the buffer *buf*.
`DatagramPacket(byte[] buf, int length, InetAddress address, int port)`
Constructs a datagram packet for sending packets with a length *length* to the specified port number on the specified host.

The *DatagramPacket* Class: Methods

DatagramPacket Methods
`public byte[] getData()`
Returns the buffer in which data has been stored.
`public InetAddress getAddress()`
Returns the IP address of the machine where the packet is being sent to or was received from.
`public int getLength()`
Returns the length of data being sent or received.
`public int getPort()`
Returns the port number on the remote host where the packet is being sent to or was received from.



MulticastSocket Class

The *MulticastSocket* Class

- A MulticastSocket is a (UDP) DatagramSocket, with additional capabilities for joining "groups" of other multicast hosts on the internet.
  - Useful for applications that implement group communication
- IP addresses for a multicast group lies within the range 224.0.0.0 to 239.255.255.255
  - Address 224.0.0.0 is reserved and should not be used

MulticastSocket Constructors
`MulticastSocket(int port)`
Creates a multicast socket bound to the given port number.

The *MulticastSocket* Class: Methods

***MulticastSocket* Methods**
`public void joinGroup(InetAddress mcastaddr)`
Join a multicast group on the specified address.
`public void leaveGroup(InetAddress mcastaddr)`
Leave a multicast group on the specified address.
`public void send(DatagramPacket p)`
An inherited method from the *DatagramSocket* class. Sends *p* from this socket.

The *MulticastSocket* Class

- Sending a message to a group
  - Should be a member of the multicast group by using the *joinGroup* method
  - Use the *send* method
  - Once done, can use the *leaveGroup* method

- The *send* method
  - Need to pass a *DatagramPacket* object

MulticastSocket Class: Server Example

```
1  import java.net.*;
2  public class ChatServer {
3     public static void main(String args[])
4                                  throws Exception {
5        MulticastSocket server =
6                        new MulticastSocket(1234);
7        InetAddress group =
8                        InetAddress.getByName("234.5.6.7");
9        //getByName- returns IP address of given host
10       server.joinGroup(group);
11       boolean infinite = true;
       /* Continually receives data and prints them */
      while(infinite) {
         byte buf[] = new byte[1024];
         DatagramPacket data =
                  new DatagramPacket(buf, buf.length);
         server.receive(data);
         String msg =
                  new String(data.getData()).trim();
         System.out.println(msg);
      }
      server.close();
   }
}
```

MulticastSocket Class: Client Example

```
1  import java.net.*;
2  import java.io.*;
3  public class ChatClient {
4     public static void main(String args[])
5                                  throws Exception {
6        MulticastSocket chat = new MulticastSocket(1234);
7        InetAddress group =
8                        InetAddress.getByName("234.5.6.7");
9        chat.joinGroup(group);
10       String msg = "";
         System.out.println("Type a message for
                              the server:");
         BufferedReader br = new BufferedReader(new
                        InputStreamReader(System.in));
         msg = br.readLine();
         DatagramPacket data = new
                  DatagramPacket(msg.getBytes(), 0,
                              msg.length(), group, 1234);
         chat.send(data);
         chat.close();
     }
  }
```

Lab:

Exercise 2: Multicast Client nd Server 1026_javase_networking.zip

Learn with Passion!
JPassion.com