

# 10 Networking



# Topics

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



# Basic Concepts on Networking

- The Internet
  - A global network of computers connected together in various ways
  - Remains functional despite of diversity of hardware and software connected together
    - Possible through communication standards defined and conformed to
    - Guarantee compatibility and reliability of communication



# Basic Concepts on Networking: IP Address

- Logically similar to the traditional mailing address
  - An address uniquely identifies a particular object
- Each computer connected to the Internet has a unique IP address
- A 32-bit number used to uniquely identify each computer connected to the Internet
  - 192.1.1.1
  - docs.rinet.ru



# Basic Concepts on Networking: 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



# Basic Concepts on Networking: 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



# Basic Concepts on Networking: 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



# Basic Concepts on Networking: 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 of service must be known before information can be transferred





# Basic Concepts on Networking: 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



# Basic Concepts on Networking: 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
  - 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

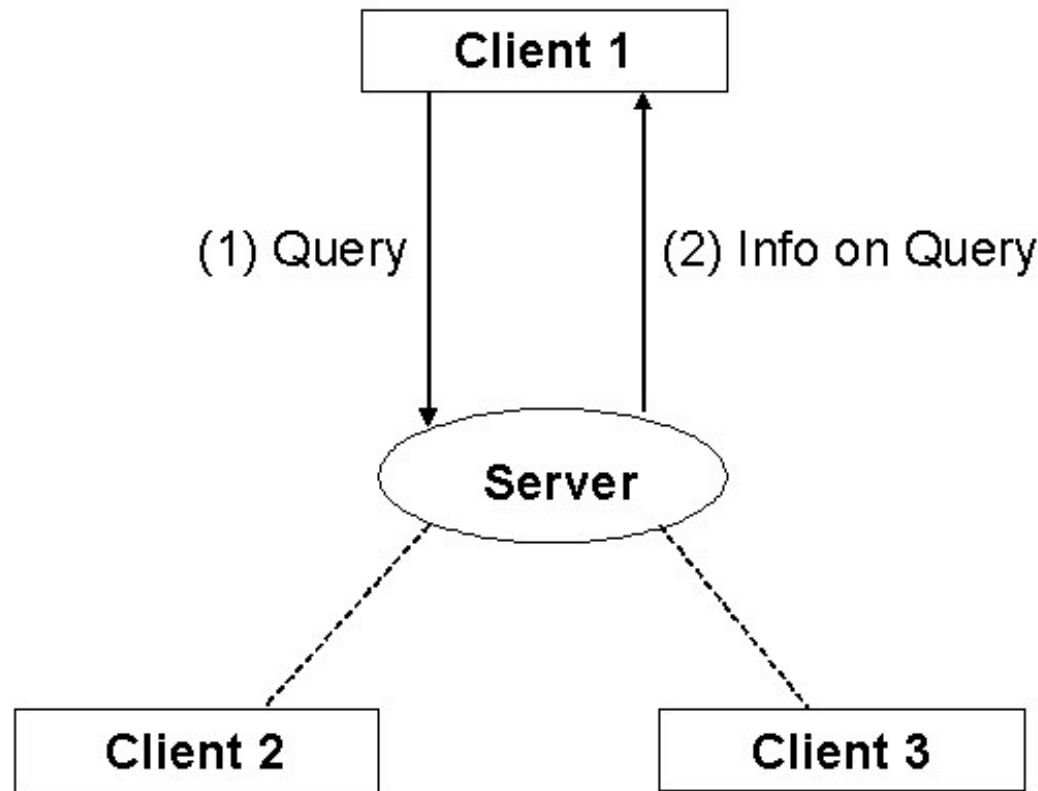


# Basic Concepts on Networking: The Client/Server Paradigm

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



# Basic Concepts on Networking: The Client/Server Paradigm



# Basic Concepts on Networking: Sockets

- Definitions:
  - 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



# The Java Networking Package

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



# The *ServerSocket* Class

- Provides the basic functionalities of a server
- Has four constructors

## ***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 based on the *backlog* parameter.



# The *ServerSocket* Class: Methods

<b><i>ServerSocket Methods</i></b>	
<code>public Socket accept()</code>	
Causes the server to wait and listen for client connections, then accept them.	
<code>public void close()</code>	
Closes the server socket. Clients can no longer connect to the server unless it is opened again.	
<code>public int getLocalPort()</code>	
Returns the port on which the socket is bound to.	
<code>public boolean isClosed()</code>	
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 server = null;
6         Socket client;
7         try {
8             server = new ServerSocket(1234);
9             //1234 is an unused port number
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             client = server.accept();
18             OutputStream clientOut =
19                 client.getOutputStream();
20             PrintWriter pw =
21                 new PrintWriter(clientOut, true);
22             InputStream clientIn =
23                 client.getInputStream();
24             BufferedReader br = new BufferedReader(new
25                 InputStreamReader(clientIn));
26             pw.println(br.readLine());
27         } catch (IOException ie) {}
28     }
```



# The *Socket* Class

- Implements a client socket
- Has eight constructors
  - Two of which are already deprecated

<b><i>Socket Constructors</i></b>
<code>Socket(String host, int port)</code>
Creates a client socket that connects to the given port number on the specified host.
<code>Socket(InetAddress address, int port)</code>
Creates a client socket that connects to the given port number at the specified IP address.



# The *Socket* Class: Methods

<b><i>Socket Methods</i></b>	
<code>public void close()</code>	
Closes the client socket.	
<code>public InputStream getInputStream()</code>	
Retrieves the input stream associated with this socket.	
<code>public OutputStream getOutputStream()</code>	
Retrieves the output stream associated with this socket.	
<code>public InetAddress getInetAddress()</code>	
Returns the IP address to which this socket is connected	
<code>public int getPort()</code>	
Returns the remote port to which this socket is connected.	
<code>public boolean isClosed()</code>	
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 }
```



# The *MulticastSocket* Class

- 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
- Has three constructors

<b><i>MulticastSocket Constructors</i></b>
<code>MulticastSocket(int port)</code>
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



# The *DatagramPacket* Class

- Used to deliver data through a connectionless protocol
- Problem: delivery of packets is not guaranteed
- Has six constructors

## ***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

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



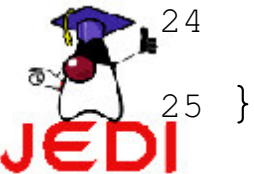
# The *MulticastSocket* Class: Server Example

```
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;
12 //continued
```



# The *MulticastSocket* Class: Server Example

```
13      /* Continually receives data and prints them */
14      while(infinite) {
15          byte buf[] = new byte[1024];
16          DatagramPacket data =
17              new DatagramPacket(buf, buf.length);
18          server.receive(data);
19          String msg =
20              new String(data.getData()).trim();
21          System.out.println(msg);
22      }
23      server.close();
24  }
25 }
```



# The *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 = "";
11 //continued...
```



# The *MulticastSocket* Class: Client Example

```
12      System.out.println("Type a message for
13                          the server:");
14      BufferedReader br = new BufferedReader(new
15                          InputStreamReader(System.in));
16      msg = br.readLine();
17      DatagramPacket data = new
18                          DatagramPacket(msg.getBytes(), 0,
19                          msg.length(), group, 1234);
20      chat.send(data);
21      chat.close();
22  }
23 }
```





# Summary

- Basic Concepts on Networking
  - IP Address
  - Protocol
  - Ports
  - The Client/Server Paradigm
  - Sockets



# Summary

- The Java Networking Package
  - *ServerSocket*
  - *Socket*
  - *MulticastSocket*
  - *DatagramPacket*

