Networking

About This Lecture

- Purpose
  To learn Java networking programming
- What You Will Learn
  - Socket-based communication and client/server computing
  - Server for multiple clients
  - Send and receive objects on the network
  - Develop applets communicating with servers
  - Class URL and class JEditorPane
  - Retrieve files from the network

Client/Server computing

- A server runs as a daemon and keeps listening on a particular port at a specific network address to allow clients to connect to it and get some service
- A client can connect to a server with the published port number and network address of the server
- Different client connection sessions to the server should not interfere with each other
- Client/server model supports specialization in computing
  - The server may be in charge of a database
  - The server may be a web server

TCP/IP

- Each computing device on the Internet has a unique IP (Internet Protocol) address of form 4 integers in range 0-255, separated by "."
  - 132.205.44.61
- TCP (Transmission Control Protocol) is a protocol to allow two computing devices on a network to communicate at data transmission level
- TCP supports reliable and persistent connection. Many parallel TCP connections between 2 parties can be established, sharing network bandwidth. Successive data transmissions on the same TCP connection will arrive at destination in order
- TCP applied on the Internet is called TCP/IP

UDP

- UDP (User Datagram Protocol) is a different network data transmission protocol
- UDP breaks message in fixed-size packets, and different packets of a message may arrive at the destination along different paths and in different order
- UDP data transmission is not reliable; the user code must take care of retransmission if necessary
- UDP is more efficient in terms of network resources
- Most Internet applications use TCP/IP

World Wide Web (WWW)

- A web server on the Internet is a server dedicated to honor client requests for static HTML pages and related multimedia files
- A user uses a web browser as his network client to connect to a web server to download and read web pages

HTTP Protocol

- A web server and its clients communicate based on HTTP (HyperText Transfer Protocol) on top of TCP/IP
- HTTP protocol is basically stateless. A web server does not store client information across multiple client connections

URL (Uniform Resource Locator)

- Each computing resource in HTTP domain has a unique URL address like
  - http://www.cs.concordia.ca
  - http://www.cs.concordia.ca/index.html
- URL names are registered through the Internet administration authorities, and mapped to IP addresses through DNSs (Domain Name Servers)

Socket and Connection (1/3)

- A socket is an abstraction of a network connection point. A socket runs on a specific port of the host machine
- A port number is any integer. It identifies the TCP service on the socket
- Port numbers 0-1023 are reserved for privileged processes (email on 25, web server on 80)
- A server must first create a server socket
  - ServerSocket svrSocket = new ServerSocket(portNbr);
  - Using a port already in use will cause java.net.BindException runtime exception

Socket and Connection (2/3)

- A client then can connect to the server by
  - Socket connectToServer = new Socket(ServerNameString, portNbr)
  - ServerNameString is a string representing server's URL or IP address
    - Socket c = new Socket("www.cs.concordia.ca", 80);
    - Socket c = new Socket("132.205.44.61", 80);
    - Socket c = new Socket("localhost", 80);
      - Server runs on the same machine
  - The new client socket will use a randomly chosen free port on the client machine for its communication with the server

Socket and Connection (3/3)

- A client connection request will be processed by the server with
  - Socket connectToClient = svrSocket.accept();
    - accepet() will wait for client connection
  - The socket will use a randomly chosen port on the server to communicate with the client
- The client and server can then create input/output streams to communicate with their partners
  - For client
    - InputStream isFromServer = connectToServer.getInputStream();
    - OutputStream osToServer = connectToServer.getOutputStream();
  - Wrap primitive streams with higher-level streams for easier data communication

A Simple Client/Server Example

- A client sends the radius of a circle
- The server returns with the area of the circle
- This example server only supports one client a time

Server.java (1/3)

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

public class Server  {
  public static void main(String[] args)  {
    try   {
      // Create a server socket
      ServerSocket serverSocket = new ServerSocket(8000);
      // Start listening for connections on the server socket
      Socket connectToClient = serverSocket.accept();
      // Create a buffered reader stream to get data from the client
      BufferedReader isFromClient = new BufferedReader(new
        InputStreamReader(connectToClient.getInputStream()));
```


Server.java (2/3)

```
      // Create a buffered writer stream to send data to the client
      PrintWriter osToClient = new PrintWriter(
          connectToClient.getOutputStream(), true);
      // Continuously read from the client and process it,
      // and send result back to the client
      while (true)   {
        // Read a line and create a string tokenizer
        StringTokenizer st = new StringTokenizer
          (isFromClient.readLine());

        // Convert string to double
        double radius = new Double(st.nextToken()).doubleValue();
        // Display radius on console
        System.out.println("radius received from client: " + radius);
```


Server.java (3/3)

```
        // Compute area
        double area = radius*radius*Math.PI;
        // Send the result to the client
        osToClient.println(area);
        // Print the result to the console
        System.out.println("Area found: "+area);
      }
    }
    catch(IOException ex)  {
      System.err.println(ex);
    }
  }
}
```


Client.java (1/3)

```
import java.io.*;
import java.net.*;
import java.util.*;
public class Client {
  public static void main(String[] args) {
    try {
      // Create a socket to connect to the server
      Socket connectToServer = new Socket("localhost", 8000);
      // Create a buffered input stream to receive data
      // from the server
      BufferedReader isFromServer = new BufferedReader(
          new InputStreamReader(connectToServer.getInputStream()));
      // Create a buffered output stream to send data to the server
      PrintWriter osToServer =
          new PrintWriter(connectToServer.getOutputStream(), true);
```


Client.java (2/3)

```
      // Continuously send radius and receive area from the server
      while (true) {
        // Read a radius from an input dialog
        Strint s = JOptionPane.showInputDialog("Please enter a radius");
        double radius = Double.parseDouble(s);

        // Send the radius to the server
        osToServer.println(radius);

        // Get area from the server
        StringTokenizer st = new StringTokenizer(
          isFromServer.readLine());

        // Convert string to double
        double area = new Double(
          st.nextToken ()).doubleValue();
```


Client.java (3/3)

```
        // Print area on the console
        System.out.println("Area received from the server is " + area);
      }
    }
    catch (IOException ex)  {
      System.err.println(ex);
    }
  }
}
```

Serving Multiple Clients (Unlimited)

- For a server to serve unlimited number of clients, use the following scheme

```
while (true) {
    Socket connectToClient = serverSocket.accept();
    Thread thread = new ThreadClass(connectToClient);
    thread.start();
}
```

- Each client will be served by a new thread
- The socket connectToClient is passed to the new thread as an argument to the thread constructor

Serving Multiple Clients (Limited)

- For a server to serve limited number of clients, use the following scheme

```
ThreadGroup g = new ThreadGroup("serving clients");
while (true) {
    if (g.activeCount() >= MAX_THREAD_LIMIT) {
        try { Thread.sleep(1000); }  // 1000 can be adjusted
        catch (InterruptedException e) {}
        continue;
    }
    Socket connectToClient = serverSocket.accept();
    Thread thread = new Thread(g, new
        ThreadClass(connectToClient));
    thread.start();
}
```


InetAddress for Client Information

- Servers can get an InetAddress object from a socket connecting to a client, and then get client's host name and IP address from the InetAddress object

```
InetAddress clientInetAddress =
    connectToClieint.getInetAddress();
System.out.println("Client's host name is " +
    clientInetAddress.getHostName());
System.out.println("Client's IP address is " +
    clientInetAddress.getHostAddress());
```


Applet Client

- Due to security constraints, an applet can only connect to the host from which it was loaded
- An applet can use class Applet's methods
  - getCodeBase() to return an URL object for the location where the applet class file resides
  - getDocumentBase() to return an URL object for the location where the html file hosting the applet resides
- We can then use URL method getHost() to get the IP address of the server
  - Socket s = new Socket(getCodeBase().getHost(), 8000);

Send/Receive Objects on the Internet

- Make sure that the object class implements interface Serializable (all descendants of java.Component do)
- Wrap up basic I/O streams with object streams

```
ObjectInputStream in = new
    ObjectInputStream(socket.getInputStream());
ObjectClass x = (ObjectClass)in.readObject();
ObjectOutputStream out = new
    ObjectOutputStream(socket.getOutputStream());
out.wrtiteObject(object)
```


Class java.net.URL

- Construction

```
try {
  URL url1 = new URL("http://www.sun.com/index.html");
  URL url2 = new URL("file:/C:\\test\\readme.txt");
}
catch (MalformedURLException e) {}
```

- Useful methods
  - String getQuery(): return the query part
  - String getPath(): return the path part
  - int getPort(): return the port number
  - String getHost(): return the host of the URL
  - String getFile(): return the file name of the URL
  - InputStream openStream(): open an InputStream to read contents of the file

Open a Web Page from Applets

- Inside the applet class, call Applet method
  AppletContext c = getAppletContext()
- Given url representing a web page, use
  c.showDocument(url);
  to display the web page in the environment. The page containing the applet becomes the preceding page

Retrieving Files from Web Servers

- Create a URL object url representing the file
- Call URL method
  - InputStream openStream() throws IOException

  to open a connection to the file's URL and return an InputStream to read its contents

View HTML Files Using JEditorPane

- Javax.swing.JEditorPane can be used to display HTML files
- Given URL object url for an HTML file, use JEditorPane method
  - void setPage(URL url) throws IOException

  to display the page
- JEditorPane generates javas.swing.event.HyperlinkEvent when a hyperlink in the editor pane is clicked. You can call getURL() against the event object to get the target URL, and use setPage() to reset the content of the editor pane

WebBrowser.java (1/5)

```
import java.net.URL;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import java.applet.*;
import java.io.*;
public class WebBrowser extends JApplet
    implements ActionListener, HyperlinkListener {
  // Editor pane to view HTML files
  private JEditorPane jep = new JEditorPane();
  //Label for URL
  private JLabel jlblURL = new JLabel("URL");
  // Text field for entering URL
  private JTextField jtfURL = new JTextField();
```


WebBrowser.java (2/5)

```
// Initialize the applet
public void init() {
  // Create a panel jpURL to hold the label and text field
  JPanel jpURL = new JPanel();
  jpURL.setLayout(new BorderLayout());
  jpURL.add(jlblURL, BorderLayout.WEST);
  jpURL.add(jtfURL, BorderLayout.CENTER);

  // Create a scroll pane to hold JEditorPane
  JScrollPane jspViewer = new JScrollPane();
  jspViewer.getViewport().add(jep, null);

  // Place jpURL and jspViewer in the applet
  this.getContentPane().add(jspViewer, BorderLayout.CENTER);
  this.getContentPane().add(jpURL, BorderLayout.NORTH);
```


WebBrowser.java (3/5)

```
  jep.setEditable(false);
  jep.addHyperlinkListener(this);
  jtfURL.addActionListener(this);
}

public void actionPerformed(ActionEvent e) {
  try {
    // Get the URL from text field
    URL url = new URL(jtfURL.getText().trim());

    // Display the HTML file
    jep.setPage(url);
  }
  catch (IOException ex) {
    System.out.println(ex);
  }
}
```

WebBrowser.java (4/5)

```
public void hyperlinkUpdate(HyperlinkEvent e) {
  try {
    jep.setPage(e.getURL());
  }
  catch (IOException ex) {
    System.out.println(ex);
  }
}
```


WebBrowser.java (5/5)

```
public static void main(String[] args) {
  JFrame frame = new JFrame("Web Browser");
  WebBrowser applet = new WebBrowser();
  frame.getContentPane().add(applet, BorderLayout.CENTER);
  applet.init();
  applet.start();
  frame.setSize(600, 600);
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  frame.setVisible(true);
 }
}
```


Run WebBrowser as Application

WebBrowser.html

```
<html>
<applet code = "WebBrowser.class"
      width = 600 height = 600 codebase=.>
</applet>
</html>
```


Security Check

- To run WebBrowser.java as an applet through appletviewer in JDK 1.2.2 or above, you have to take the following steps to cross security checks:
  - Create a policy file named *java.policy* containing

```
grant {
  // Allow everything for now
  permission java.security.AllPermission;
};
```

  - Run the applet with

```
appletviewer –J-Djava.security.policy=java.policy
  WebBrowser.html
```