## Java Networking -- Socket

- Server socket class: `ServerSocket`
  - wait for requests from clients.
  - after a request is received, a client socket is generated.
- Client socket class: `Socket`
  - an endpoint for communication between two apps/applets.
  - obtained by
    - contacting a server
    - generated by the server socket
- Communication is handled by input/output streams.
  - `Socket` provides an input and an output stream.

## A Simple Echo Server

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

public class EchoServer {
  public static void main(String[] args) {
    try {
      ServerSocket s = new ServerSocket(8008);
      while (true) {
        Socket incoming = s.accept();
        BufferedReader in
          = new BufferedReader(
                new InputStreamReader(
                    incoming.getInputStream()));
        PrintWriter out
          = new PrintWriter(
                new OutputStreamWriter(
                    incoming.getOutputStream()));
```

## A Simple Echo Server (cont'd)

```
        out.println("Hello! ....");
        out.println("Enter BYE to exit.");
        out.flush();
        while (true) {
          String str = in.readLine();
          if (str == null) {
            break; // client closed connection
          } else {
            out.println("Echo: " + str);
            out.flush();
            if (str.trim().equals("BYE"))
              break;
          }
        }
        incoming.close();
      }
    } catch (Exception e) {}
  }
}
```

## Test the EchoServer with `Telnet`

Use `telnet` as a client.

```
venus% telnet saturn 8008
Trying 140.192.34.63 ...
Connected to saturn.
Escape character is '^]'.
Hello! This is the Java EchoServer.
Enter BYE to exit.
Hi, this is from venus
Echo: Hi, this is from venus
BYE
Echo: BYE
Connection closed by foreign host.
```

# A Simple Client

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

public class EchoClient {

  public static void main(String[] args) {
    try {
      String host;
      if (args.length > 0) {
        host = args[0];
      } else {
        host = "localhost";
      }
      Socket socket = new Socket(host, 8008);
```

# A Simple Client (cont'd)

(class `EchoClient` continued.)

```
      BufferedReader in
        = new BufferedReader(
              new InputStreamReader(
                  socket.getInputStream()));
      PrintWriter out
        = new PrintWriter(
              new OutputStreamWriter(
                  socket.getOutputStream()));
      // send data to the server
      for (int i = 1; i <= 10; i++) {
        System.out.println("Sending: line " + i);
        out.println("line " + i);
        out.flush();
      }
      out.println("BYE");
      out.flush();
```

# A Simple Client (cont'd)

(class `EchoClient` continued.)

```
      // receive data from the server
      while (true) {
        String str = in.readLine();
        if (str == null) {
          break;
        } else {
          System.out.println(str);
        }
      }
    } catch (Exception e) {}
  }
}
```

# Multi-Threaded Echo Server

- To handle multiple requests simultaneously.
- In the `main()` method, spawn a thread for each request.

```
public class MultiEchoServer {

  public static void main(String[] args) {
    try {
      ServerSocket s = new ServerSocket(8009);
      while (true) {
        Socket incoming = s.accept();
        new ClientHandler(incoming).start();
      }
    } catch (Exception e) {}
  }

}
```

# Client Handler

```
public class ClientHandler extends Thread {

  protected Socket incoming;

  public ClientHandler(Socket incoming) {
    this.incoming = incoming;
  }
  public void run() {
    try {
      BufferedReader in
        = new BufferedReader(
              new InputStreamReader(
                  incoming.getInputStream()));
      PrintWriter out
        = new PrintWriter(
              new OutputStreamWriter(
                  incoming.getOutputStream()));
```

# Client Handler (cont'd)

```
      out.println("Hello! ...");
      out.println("Enter BYE to exit.");
      out.flush();
      while (true) {
        String str = in.readLine();
        if (str == null) {
          break;
        } else {
          out.println("Echo: " + str);
          out.flush();
          if (str.trim().equals("BYE"))
            break;
        }
      }
      incoming.close();
    } catch (Exception e) {}
  }
}
```

# Visitor Counter

- A server and an applet.
- The server keeps the visitor count in a file, and sends the count to clients when requested.
- No need for spawning thread, since only need to transmit an integer.
- Read and write files.

# Visitor Counter Server

```
public class CounterServer {

  public static void main(String[] args) {
    System.out.println("CounterServer started.");
    int i = 1;
    try {
      // read count from the file
      InputStream fin =
          new FileInputStream("Counter.dat");
      DataInputStream din =
          new DataInputStream(fin);
      i = din.readInt() + 1;
      din.close();
    } catch (IOException e) {}
```

# Visitor Counter Server (cont'd)

(class `CountServer` continued.)

```
    try {
      ServerSocket s = new ServerSocket(8190);
      while (true) {
        Socket incoming = s.accept();
        DataOutputStream out
          = new DataOutputStream(
                  incoming.getOutputStream());
        System.out.println("Count: " + i);
        out.writeInt(i);
        incoming.close();
```

# Visitor Counter Server (cont'd)

(class `CountServer` continued.)

```
        OutputStream fout =
            new FileOutputStream("Counter.dat");
        DataOutputStream dout =
            new DataOutputStream(fout);
        dout.writeInt(i);
        dout.close();
        out.close();
        i++;
      }
    } catch (Exception e) {}
    System.out.println("CounterServer stopped.");
  }

}
```

# Counter Applet

```
public class Counter extends Applet {

  protected int count = 0;
  protected Font font =
        new Font("Serif", Font.BOLD, 24);

  public void init() {
    URL url = getDocumentBase();
    try {
      Socket t = new Socket(url.getHost(), 8190);
      DataInputStream in =
        new DataInputStream(t.getInputStream());
      count = in.readInt();
    } catch (Exception e) {}
  }
```

# Counter Applet (cont'd)

(class `CountServer` continued.)

```
  public void paint(Graphics g) {
    int x = 0, y = font.getSize();
    g.setColor(Color.green);
    g.setFont(font);
    g.drawString("You are visitor: " + count,
                 x, y);
  }
}
```

## Broadcast Echo Server

- To handle multiple requests simultaneously.
- Broadcast messages received from any client to all active clients.
- Need to keep track of all active clients.

## Broadcast Echo Server (cont'd)

```
public class BroadcastEchoServer {
  static protected Set activeClients =
                                    new HashSet();
  public static void main(String[] args) {
    int i = 1;
    try {
      ServerSocket s = new ServerSocket(8010);
      while (true) {
        Socket incoming = s.accept();
        BroadcastClientHandler newClient =
          new BroadcastClientHandler(incoming, i++);
        activeClients.add(newClient);
        newClient.start();
      }
    } catch (Exception e) {}
  }
}
```

## Broadcast Client Handler

```
public class BroadcastClientHandler
    extends Thread {

  protected Socket incoming;
  protected int id;
  protected BufferedReader in;
  protected PrintWriter out;

  public synchronized void
  sendMessage(String msg) {
    if (out != null) {
      out.println(msg);
      out.flush();
    }
  }
```

## Broadcast Client Handler (cont'd)

```
  public BroadcastClientHandler(Socket incoming,
                                int id) {
    this.incoming = incoming;
    this.id = id;
    try {
      if (incoming != null) {
        in = new BufferedReader(
               new InputStreamReader(
                 incoming.getInputStream()));

        out = new PrintWriter(
                new OutputStreamWriter(
                  incoming.getOutputStream()));
        }
      } catch (Exception e) {}
    }
```

# Broadcast Client Handler (cont'd)

```
public void run() {
  if (in != null &&
      out != null) {
    sendMessage("Hello! ...");
    sendMessage("Enter BYE to exit.");
    try {
      while (true) {
        String str = in.readLine();
        if (str == null) {
          break;
        } else {
          // echo back to this client
          sendMessage("Echo: " + str );
          if (str.trim().equals("BYE")) {
            break;
          } else {
```

# Broadcast Client Handler (cont'd)

```
            // broadcast to other active clients
            Iterator iter =
         BroadcastEchoServer.activeClients.iterator();
            while  (iter.hasNext()) {
              BroadcastClientHandler t =
                (BroadcastClientHandler) iter.next();
              if (t != this)
                t.sendMessage("Broadcast(" +
                              id + "): " + str);

        }}}}
        incoming.close();
        // this client is no longer active
        BroadcastEchoServer.activeClients.remove(this);
      } catch (IOException e) {}
    }
  }
}
```