Network

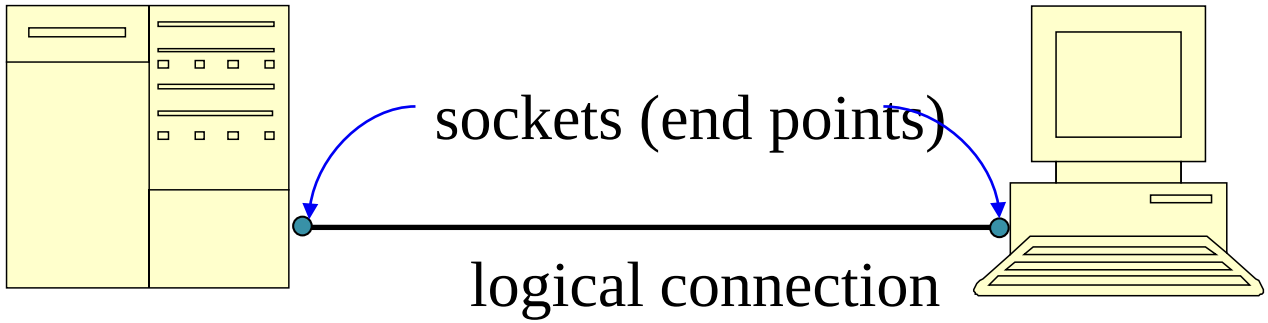
Dr. Jens Bennedsen,
Aarhus University, School of Engineering
Aarhus, Denmark
jbb@ase.au.dk

Outline

- Socket programming
- If we have the time: Remote method invocation (RMI)

Socket Programming

- Sockets
  - Ends points of two-way communications, i.e., logical connections between hosts
  - Can be used to send and receive data
  - Supported by most languages and platforms
  - Server vs. client sockets

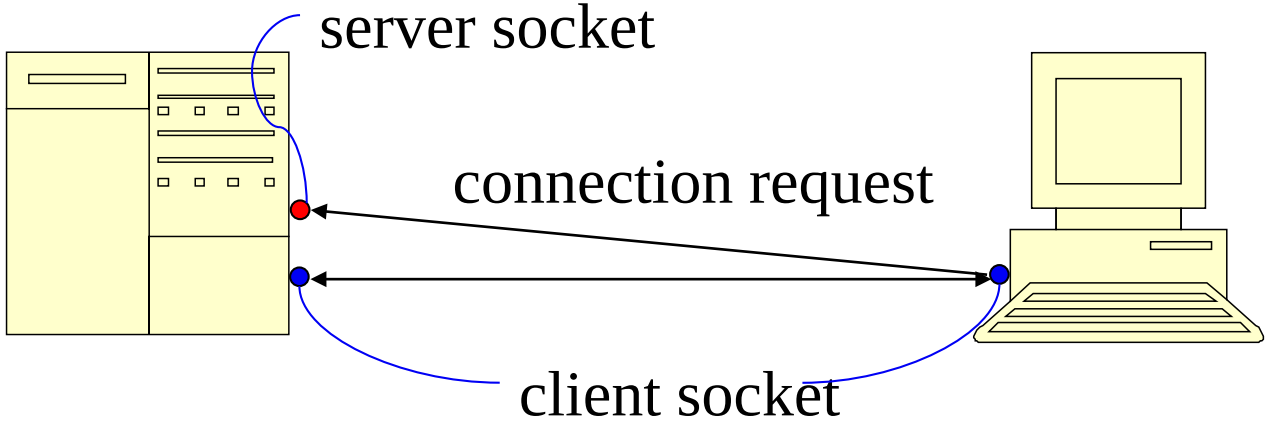


Socket Programming (Cont.)

- Server sockets
  - Wait for requests to come in over the network
  - Implemented by java.net.ServerSocket class
- Client sockets
  - Used to send and receive data
  - Can be thought of as a pair of input and output streams
  - Implemented by java.net.Socket class

Server vs. Client Sockets

- Server socket: waiting for connection requests
- Client socket: transmission of data



Server Sockets

- Creating and using server sockets

Constructors
`ServerSocket(int port)` `ServerSocket(int port, int backlog)`

Methods	Description
`accept()`	Waits for a connection request and returns a Socket
`close()`	Stops waiting for requests from clients

Server Sockets (Cont.)

- Usage pattern

```
try {
   ServerSocket server = new ServerSocket(8888);
   while (true) {
       Socket incoming = server.accept(); // obtain a client socket
       // handle client request by reading from and writing to the socket …
   }
} catch (IOException e) {
   // handle exception on creating a server socket
}
```

Client Sockets

- Creating client sockets
  - On the client side, Socket(String host, int port)
  - On the server side, accept() of ServerSocket
- Using client sockets

Methods	Description
`getInputStream()`	Returns an InputStream for receiving data
`getOutputStream()`	Returns an OutputStream to send data
`close()`	Closes the socket connection

Client Sockets (Cont.)

- Usage pattern

```
try {
   Socket s = new Socket("host name", 8888);
   PrintWriter out = new PrinterWriter(
       new OutputStreamWriter(s.getOutputStream()));
   BufferedReader in = new BufferedReader(
       new InputStreamReader(s.getInputStream()));
   // send and receive data by using out and in …
   in.close();
   out.close();
   s.close();
} catch (IOException e) {
   // handle exception …
}
```

Example -- A Simple Echo Server

```
import java.io.*;
import java.net.*;
public class EchoServer {
   public static void main(String[] args) {
       try {
         ServerSocket server = new ServerSocket(8008);
         while (true) {
            Socket s = server.accept();
            BufferedReader in = new BufferedReader(
                                    new InputStreamReader(
                                         s.getInputStream()));
            PrintWriter out = new PrintWriter(
                                   new OutputStreamWriter(
                                        s.getOutputStream()));
            <<handle client by using in and out – next slide>>
            s.close();
         }
       } catch (Exception e) { e.printStackTrace(); }
   }
}
```

Echo Server (Cont.)

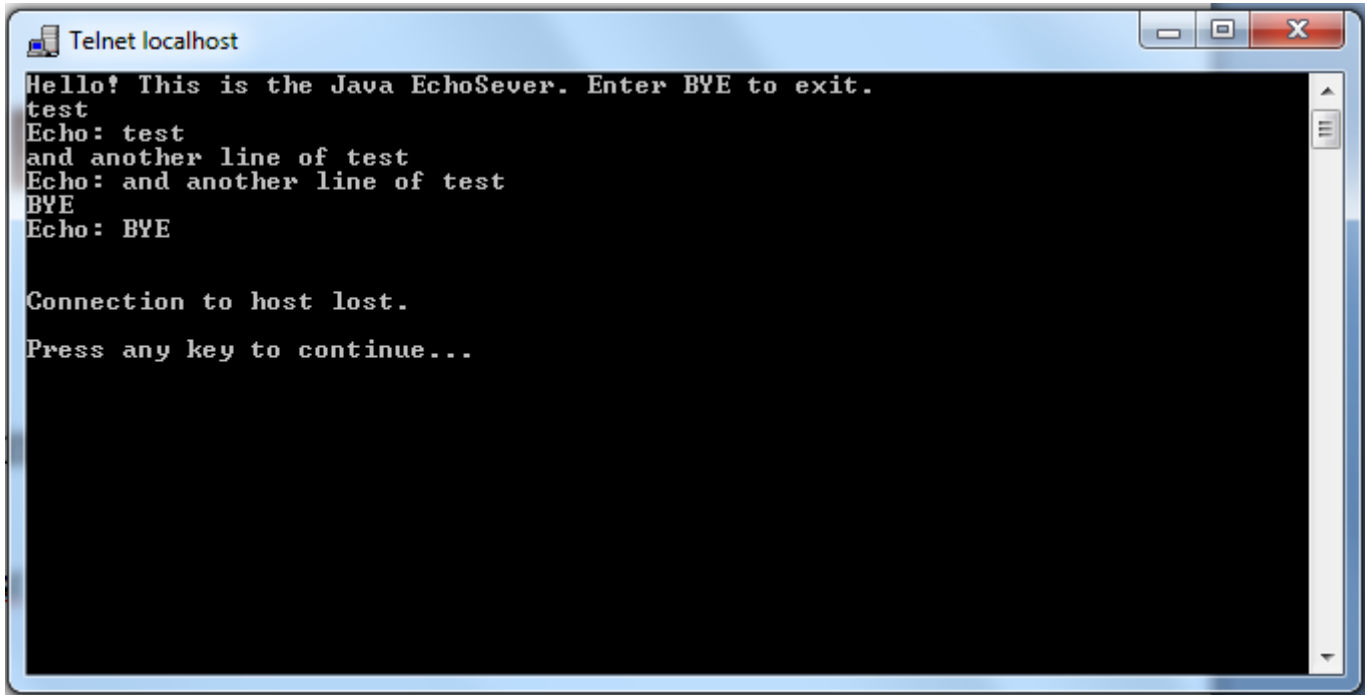
<<handle client by using in and out>>=

```
out.print(“Hello! This is the Java EchoSever. ”);
out.println(“Enter BYE to exit.”);
out.flush();
String str = null;
while ((str = in.readLine()) != null) {
   System.out.println(“Received: “ + str);
   out.println(“Echo: “ + str);
   out.flush();
   if (str.trim().equals(“BYE”)) {
      break;
   }
}
```

Testing Echo Server

Testing with telnet client

```
Microsoft Telnet> open localhost 8008
Connecting To localhost...
```



Turning on telnet in windows

How do I install Telnet?

By default, Telnet is not installed with Windows, but you can install it by following the steps below.

To install Telnet Client

1. Click the **Start** button, click **Control Panel**, click **Programs**, and then click **Turn Windows features on or off**. If you are prompted for an administrator password or confirmation, type the password or provide confirmation.
2. In the **Windows Features** dialog box, select the **Telnet Client** check box.
3. Click **OK**. The installation might take several minutes.

How do I use Telnet in Windows?

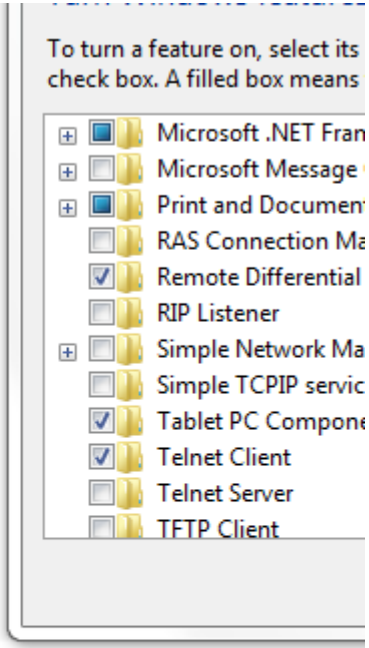
Telnet doesn't look like Windows. Why?

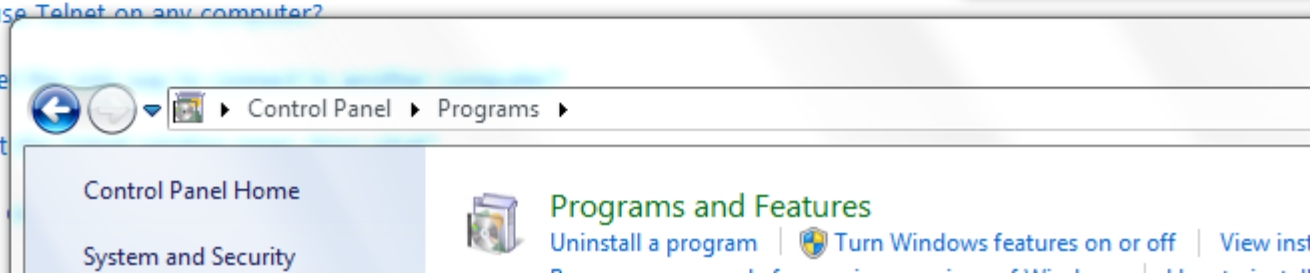
Can I use Telnet on any computer?

Is Telne

I've got

Where





A Simple Echo Client

```
import java.io.*;
import java.net.*;
public class EchoClient {
   public static void main(String[] args) {
       String host = args.length > 0 ? host = args[0] : “localhost”;
       try {
         Socket socket = new Socket(host, 8008);
         BufferedReader in = new BufferedReader(
                                 new InputStreamReader(
                                     socket.getInputStream()));
         PrintWriter out = new PrintWriter(new OutputStreamWriter(
                                              socket.getOutputStream()));
         <<send and receive data by using in and out –next slide>>
         socket.close();
       } catch (Exception e) {
         e.printStackTrace();
       }
   }
}
```

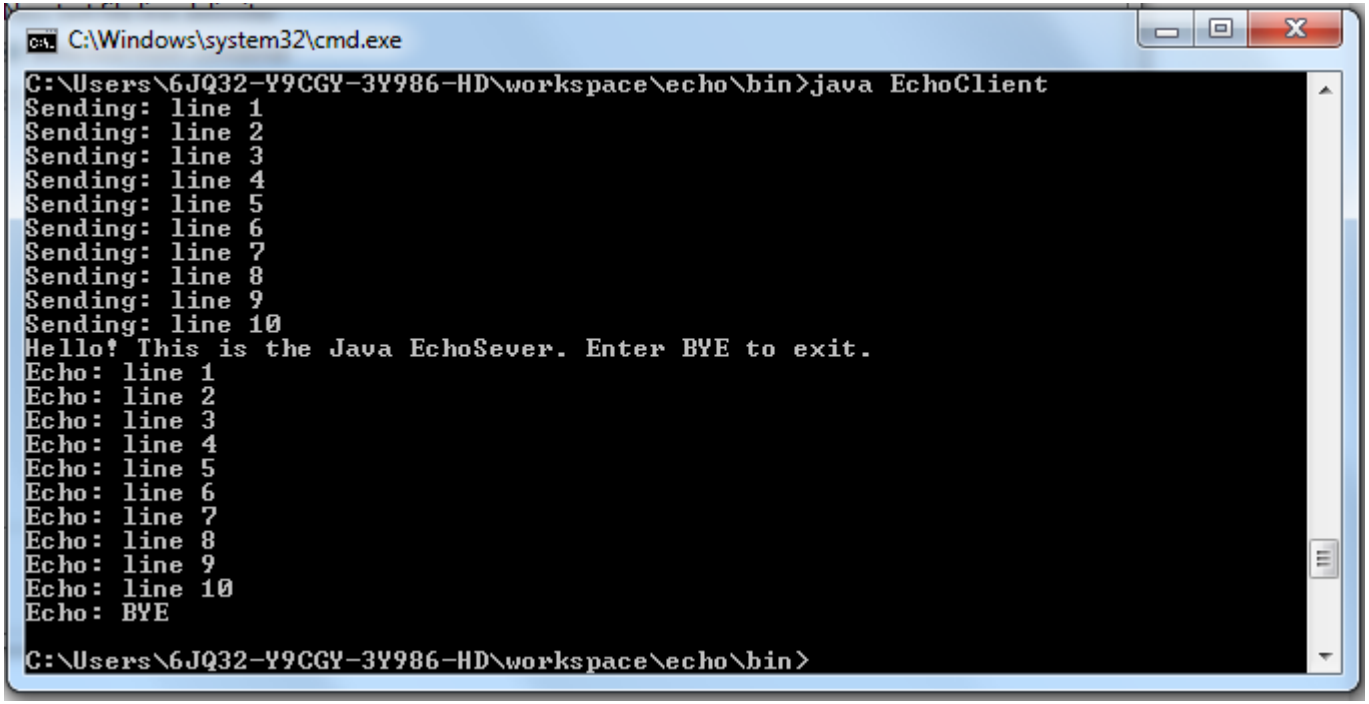
Echo Client (Cont.)

<<send and receive data by using in and out>>=

```
// send data to 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();

// receive data from server
String str = null;
while ((str = in.readLine()) != null) {
   System.out.println(str);
}
```

Testing Echo Client



Echo Server Revisited

How to support multiple clients simultaneously?

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

public class MultiEchoServer {
   public static void main(String[] args) {
       try {
         ServerSocket server = new ServerSocket(8008);
         while (true) {
            Socket s = server.accept();
            new ClientHandler(s).start();
          }
       } catch (Exception e) {
          e.printStackTrace();
       }
   }

   <<class ClientHandler>>
}
```

Multi Echo Server (Cont.)

<<class ClientHandler>>=

```
private static class ClientHandler extends Thread {
   private Socket sock;

   public ClientHandler(Socket sock) { this.sock = sock; }

   public void run() {
       BufferedReader in = new BufferedReader(new InputStreamReader(
            sock.getInputStream()));
       PrintWriter out = new PrintWriter(new OutputStreamWriter(
            sock.getOutputStream()));
       out.println("Hello! This is the Java EchoSever.\n
                    Enter BYE to exit.");
       out.flush();
       String str = null;
       while ((str = in.readLine()) != null) {
          out.println("Echo: " + str); out.flush();
          if (str.trim().equals("BYE")) { break; }
       }
   }
}
```

Outline

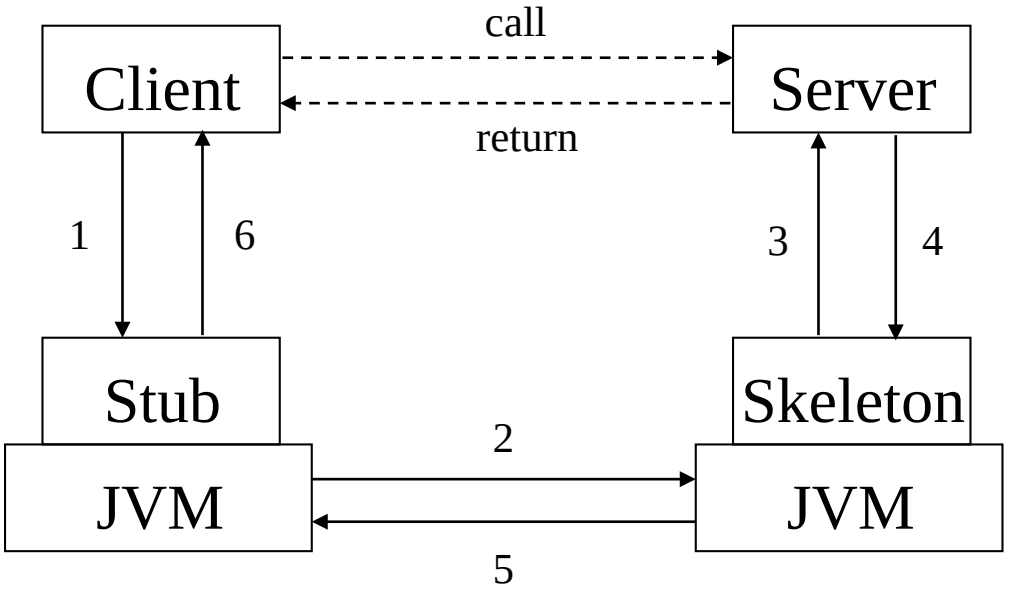
- √ Socket programming
- Remote method invocation (RMI)

Remote Method Invocation

- Why RMI?
  - In socket programming, programmers have to make explicit connections between clients and servers and manage data transmission.
  - Thus, it's hard and error-prone to write socket programs.
  - Can the connection and data transmission be managed by JVM?

What's RMI?

- Distributed programming model
  - to allow objects residing on different hosts (*remote objects*) to be manipulated as if they were all on the same host (*local objects*)
- RMI architecture



Local vs. Remote Objects

- Local objects
  - Objects accessible only within the local hosts
- Remote objects
  - Objects accessible from remote hosts
  - Instances of classes that implements a marker interface java.rmi.Remote
- Property of remote objects
  - Similar to local objects (arguments, downcasting, instanceof, etc)
  - Clients of remote objects interact with stubs
  - Passing arguments and results for RMI calls
    - Call by value for local objects (through serialization and deserialization)
    - Call by reference for remote objects

Locating Remote Objects

- RMI registry
  - Directory service mapping RMI servers (or objects) to their names
  - Server: register itself to make it available to remote clients
  - Client: locate a server by looking up an RMI registry with a URL protocol rmi, e.g.,

    rmi://host:port/name

  - The programming interface by the class java.rmi.Naming

Method	Description
`bind(name, obj)`	`Bind obj to name`
`rebind(name, obj)`	`Bind obj to name even if already bound`
`unbind(name)`	`Remove the binding`
`lookup(url)`	`Return object bound to url`
`list(url)`	`Return a list of all bindings`

Writing RMI Programs

1. Define a remote interface, e.g.,

```
public interface Service extends java.rmi.Remote {
    public void doSomething(…) throws java.rmi.RemoteException;
    // …
}
```

2. Define a service implementation class, e.g.,

```
public class ServiceProvider extends
    java.rmi.server.UniCastRemoteObject
    implements Service {
    public void doSomething(…) throws java.rmi.RemoteException {
       // …
    }
    // …
}
```

Writing RMI Programs (Cont.)

3. Create a server instance and register to an RMI registry, e.g.,

```
Service server = new ServiceProvider(…);
java.rmi.Naming.bind(name, server);
```

4. Generate the stub and skeleton classes by using the RMI compiler (rmic), e.g.,

```
% rmic ServiceProvider
```

The command produces:
ServiceProvider_Stub.class and ServiceProvider_Skel.class

Writing RMI Programs (Cont.)

5. Write a client program, e.g.,

```
java.rmi.Remote obj = java.rmi.Naming.lookup(name);
Service server = (Service) obj;
…
server.doSomething(…); // RMI call
…
```

Example -- A Simple Time Server

1. Create an interface. (in this case, `TimeService.java`).
2. Create a class that implements the interface. (in this case, `TimeServer.java`).
3. Create a client that connects to the server object using Naming.lookup() (in this case, `TimeClient.java`).
4. Compile these classes.
5. Run the RMI interface compiler on the .class file of the implementation class (in this case, you'd say "`rmic TimeServer`").
6. Start the RMI registry (on Windows say "`start rmiregistry`").
7. Start the server class ("`start java TimeServer`").
8. Run the client program ("`java TimeClient`").

A Simple TimeServer

- Remote interface, TimeService

```
public interface TimeService extends java.rmi.Remote {
    java.util.Date getTime() throws java.rmi.RemoteException;
}
```

- Server and client classes

A Server Class, TimeServer

```
import java.rmi.*;
import java.util.*;

public class TimeServer extends
java.rmi.server.UnicastRemoteObject
    implements TimeService {

    public TimeServer() throws RemoteException {}

    public Date getTime() {
       return Calendar.getInstance().getTime(); }

    public static void main(String [] args) {
        try {
            TimeServer server = new TimeServer();
            Naming.rebind("TimeServer", server);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

A Client Class, TimeClient

```
import java.rmi.*;
improt java.util.*;

public class TimeClient {

    public static void main(String [] args) {
        try {
            TimeService server = (TimeService)
                Naming.lookup( "rmi://localhost/TimeServer");
            System.out.println(server.getTime());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Compiling and Running (UNIX)

1. Compile the server and client programs, e.g.,

   % javac TimeServer.java TimeClient.java TimeService.java

2. Generates the stubs and skeletons, e.g.,

   % rmic TimeServer

3. Start the RMI registry on the server host, e.g.,

   % rmiregistry &

4. Run the server on the server host, e.g.,

   % java TimeServer &

5. Runt the client on the client host, e.g.,

   % java TimeClient

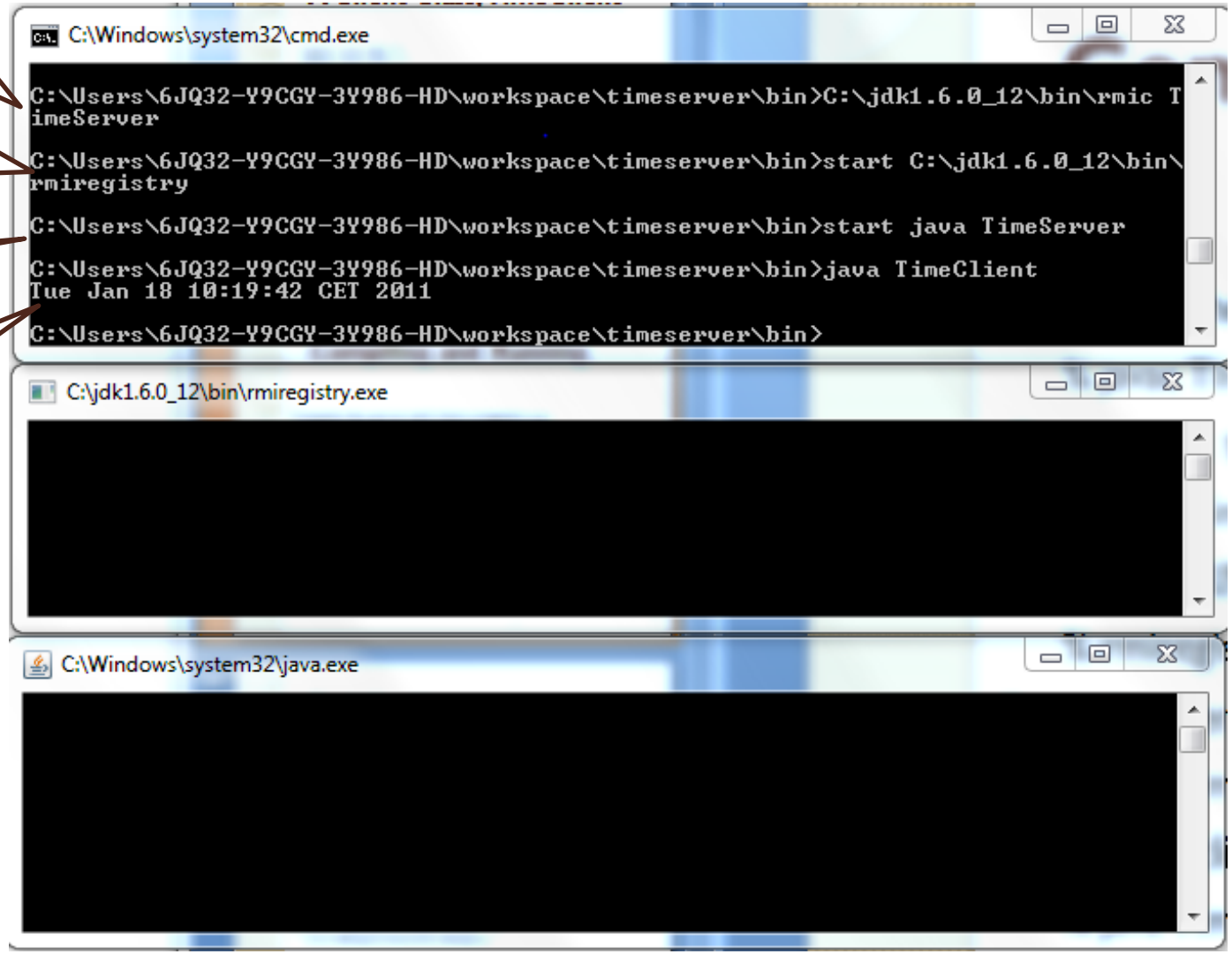
Compile and run (Win)

Create stubs

Start registry

Start server

Start client



Serialization

- What is it?
  - Process of transforming an object into a stream of bytes; the reverse process is called *deserialization*.
  - Allows objects to be saved to files or sent to remote hosts over a network (e.g., arguments to RMI calls)
- How to make objects serializable?
  - By implementing the marker interface java.io.Serializable
  - A default implementation for (de) serialization is automatically provided.
  - Can customize the process by implementing readObject() and writeObject() methods:

```
private void writeObject(java.io.ObjectOutputStream out)
    throws IOException;
private void readObject(java.io.ObjectInputStream in)
    throws IOException, ClassNotFoundException;
```

Example

- Make the following class Student serializable

```
public class Student {
    private String name;
    private int score;
    //@ private invariant 0 <= score && score <= 100;
    private char grade;
  //@ private invariant (* grade is one of ‘A’, …, ‘F’
  *);
    // …
}
```

- Answer 1:

```
public class Student implements Serializable {
    // …
}
```

Example (Cont.)

- Answer 2:

```
public class Student implements Serializable {
        // …
        private void writeObject(java.io.ObjectOutputStream out)
throws IOException {
           out.writeUTF(name);
           out.writeInt(score);
           out.writeChar(grade);
        }
   private void readObject(java.io.ObjectInputStream in)
   throws IOException, ClassNotFoundException {
       name = in.readUTF();
       score = in.readInt();
       grade = in.readChar();
   }
}
```

Example (Cont.)

- Answer 3:

```
public class Student implements Serializable {
        // …
   private void writeObject(java.io.ObjectOutputStream out)
throws IOException {
       out.writeUTF(name);
       out.writeInt(score);
   }
   private void readObject(java.io.ObjectInputStream in)
    throws IOException, ClassNotFoundException {
       name = in.readUTF();
       score = in.readInt();
       grade = calculateGrade(score);
   }
   private char calculateGrade(int score) { /* … */ }
}
```

Using Serialization

- Serializaing objects

```
ObjectOutputStream out = new ObjectOutputStream(/* … */);
Student s = new Student(/* … */);
out.writeObject(s);
// …
```

- Deserializing objects

```
ObjectInputStream in = new ObjectInputStream(/* … */);
   Object obj = in.readObject();
   Student s = (Student) obj;
   // …
```