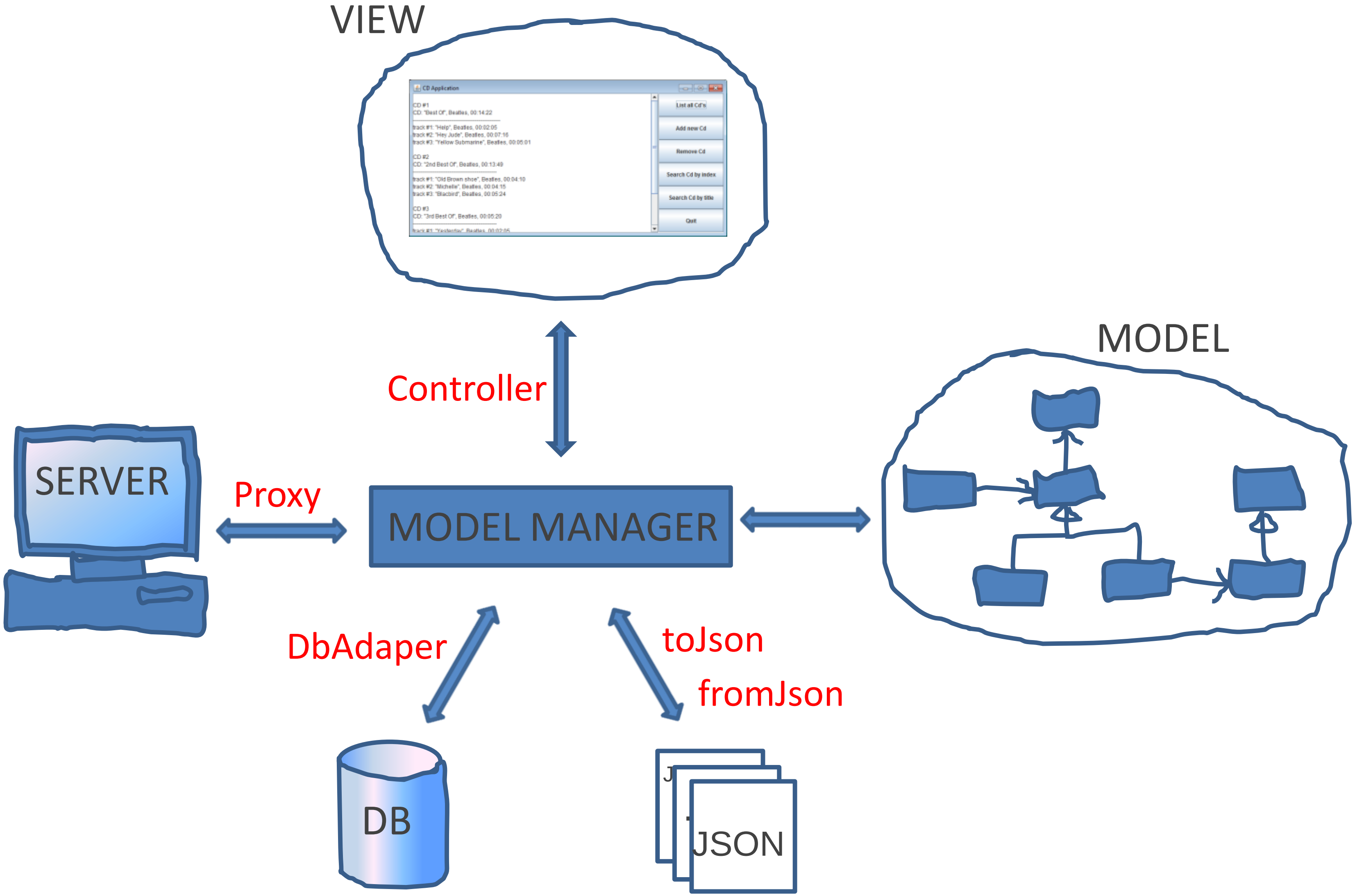
VIA University College

Software Development with UML and Java 2

Learning Objectives

- Sockets in System Design
- Sockets with JSON

System Overview



VIEW

CD Application

```
CD #1
CD: "Best Of", Beatles, 00:14:22
----------------------------------------
track #1: "Help", Beatles, 00:02:05
track #2: "Hey Jude", Beatles, 00:07:16
track #3: "Yellow Submarine", Beatles, 00:05:01

CD #2
CD: "2nd Best Of", Beatles, 00:13:49
----------------------------------------
track #1: "Old Brown shoe", Beatles, 00:04:10
track #2: "Michelle", Beatles, 00:04:15
track #3: "Blacbird", Beatles, 00:05:24

CD #3
CD: "3rd Best Of", Beatles, 00:05:20
----------------------------------------
track #1: "Yesterday", Beatles, 00:02:05
```

Controller

MODEL

SERVER

Proxy

MODEL MANAGER

DbAdaper

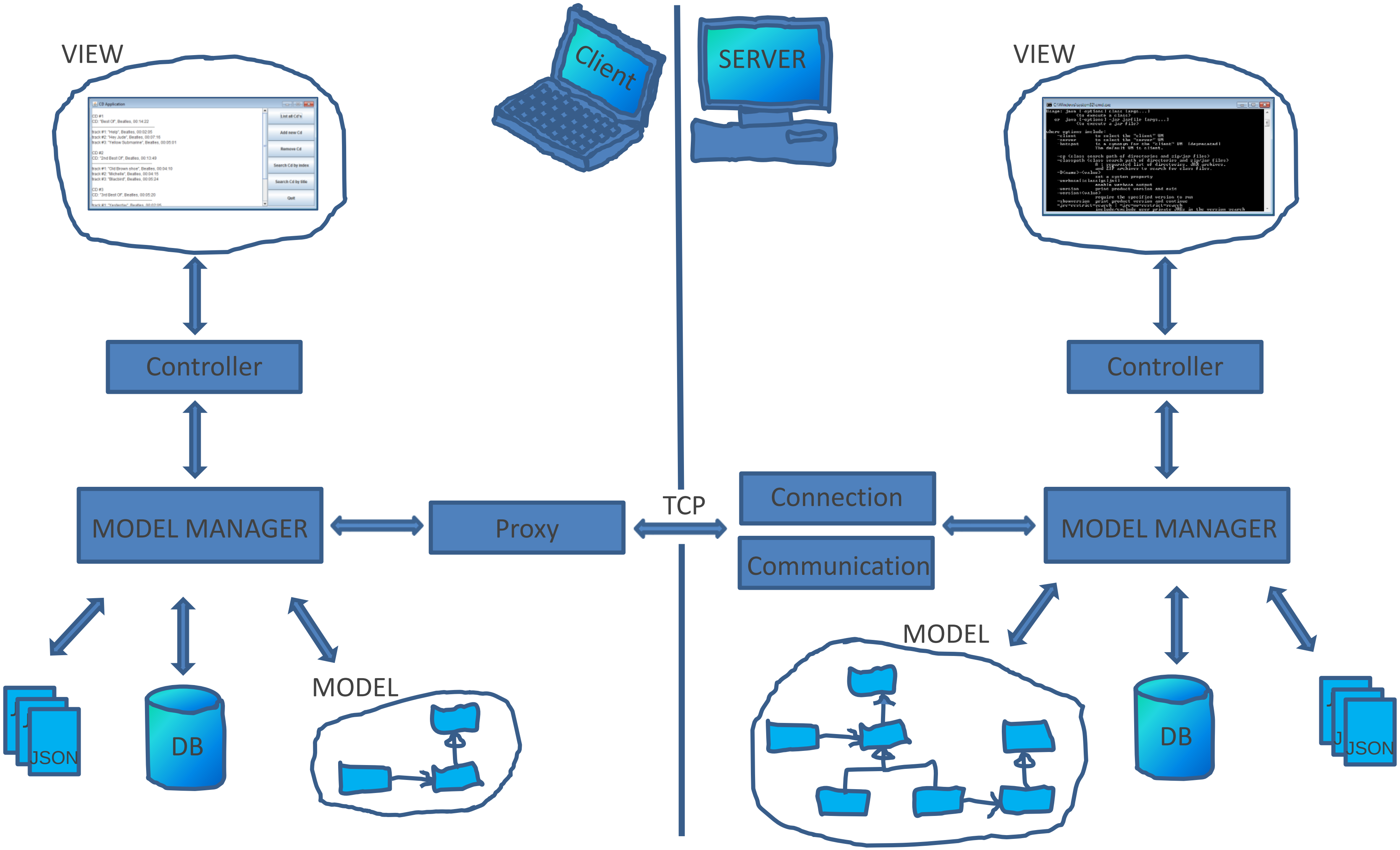
toJson

fromJson

DB

JSON

Client/Server Overview



```
CD #1
CD: "Best Of", Beatles, 00:14:22
--------------------------------
track #1: "Help", Beatles, 00:02:05
track #2: "Hey Jude", Beatles, 00:07:16
track #3: "Yellow Submarine", Beatles, 00:05:01

CD #2
CD: "2nd Best Of", Beatles, 00:13:49
--------------------------------
track #1: "Old Brown shoe", Beatles, 00:04:10
track #2: "Michelle", Beatles, 00:04:15
track #3: "Blacbird", Beatles, 00:05:24

CD #3
CD: "3rd Best Of", Beatles, 00:05:20
--------------------------------
track #1: "Yesterday", Beatles, 00:02:05
```

```
Usage: java [-options] class [args...]
           (to execute a class)
   or  java [-options] -jar jarfile [args...]
           (to execute a jar file)

where options include:
    -client       to select the "client" VM
    -server       to select the "server" VM
    -hotspot      is a synonym for the "client" VM  [deprecated]
                  The default VM is client.

    -cp <class search path of directories and zip/jar files>
    -classpath <class search path of directories and zip/jar files>
                  A ; separated list of directories, JAR archives,
                  and ZIP archives to search for class files.
    -D<name>=<value>
                  set a system property
    -verbose[:class|gc|jni]
                  enable verbose output
    -version      print product version and exit
    -version:<value>
                  require the specified version to run
    -showversion  print product version and continue
    -jre-restrict-search | -jre-no-restrict-search
                  include/exclude user private JREs in the version search
```

Some design questions for a system using sockets

1. Sockets using TCP or UDP?
2. Streaming serialized objects, strings, XML or JSON
3. Which protocol for client/server communication?
4. How to make the server stable even if clients are not following the communication protocol?
5. How to avoid program termination or data loss when a socket connection is lost
6. MVC: How to make communication with sockets transparent to Model and View (and Controller)?
7. How to work with multiple threads for socket communication?

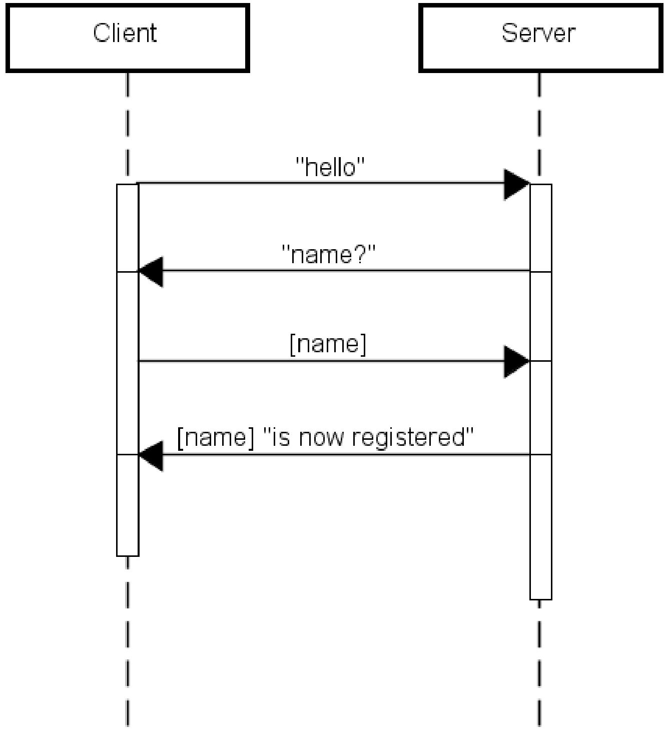
More design questions…

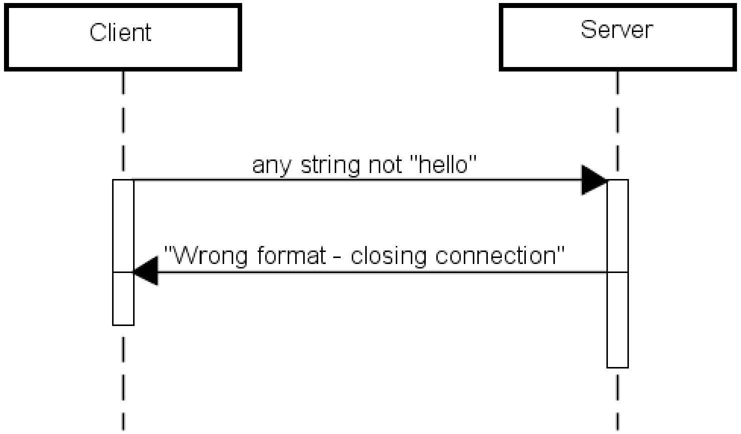
8. How to let clients be observers on a server model?
9. How to know the difference between a server reply and a message broadcasted to all clients?
10. What to store in the client model?
11. When to get values from client model and when to get from server model?
12. When to store in server model, in database, in JSON?
13. Should the server use backup e.g. JSON or database?
14. How to work with different states, e.g. a sending state and a receiving state?

Protocol Design

3. Which protocol for client/server communication?

- You have to draw the diagrams for communication flow – no matter which communication protocol you may decide to follow





JSON

- JavaScript Object Notation
  - However, JSON is a (mostly) language-independent way of specifying objects as name-value pairs
- Used to format data
- Commonly used in Web applications as a tool to describe data being sent between systems

JSON Syntax

- An *object* is an unordered set of name/value pairs
  - pairs are enclosed within braces, { }
  - There is a colon between the name and the value
  - Pairs are separated by commas
  - Example: { "name": “Bob", “number": 3 }
- An *array* is an ordered collection of values
  - The values are enclosed within brackets, [ ]
  - Values are separated by commas
  - Example: [“student", ”staff", “guest"]

XML vs JSON

- XML
  - a data format
  - a way to structure data
-
- JSON
  - a data format
  - a way to structure data

XML vs JSON

- Some Similarities:
  - They are both human readable
  - They both have very simple syntax
  - They are both hierarchical
  - They are both language independent
  - They are both supported in APIs of many programming languages
- Some Differences:
  - Syntax is different
  - JSON is less verbose
  - JSON includes arrays
  - Names in JSON must not be JavaScript reserved words
  - XML can be validated

XML-formated Example

- Book

```
<Book>
    <Title>Applying UML and patterns</Title>
    <Authors>
        <Author>Craig Larman</Author>
    </Authors>
    <Date>2004</Date>
    <Publisher>Pearson</Publisher>
</Book>
```

JSON-formatted

- Same book data

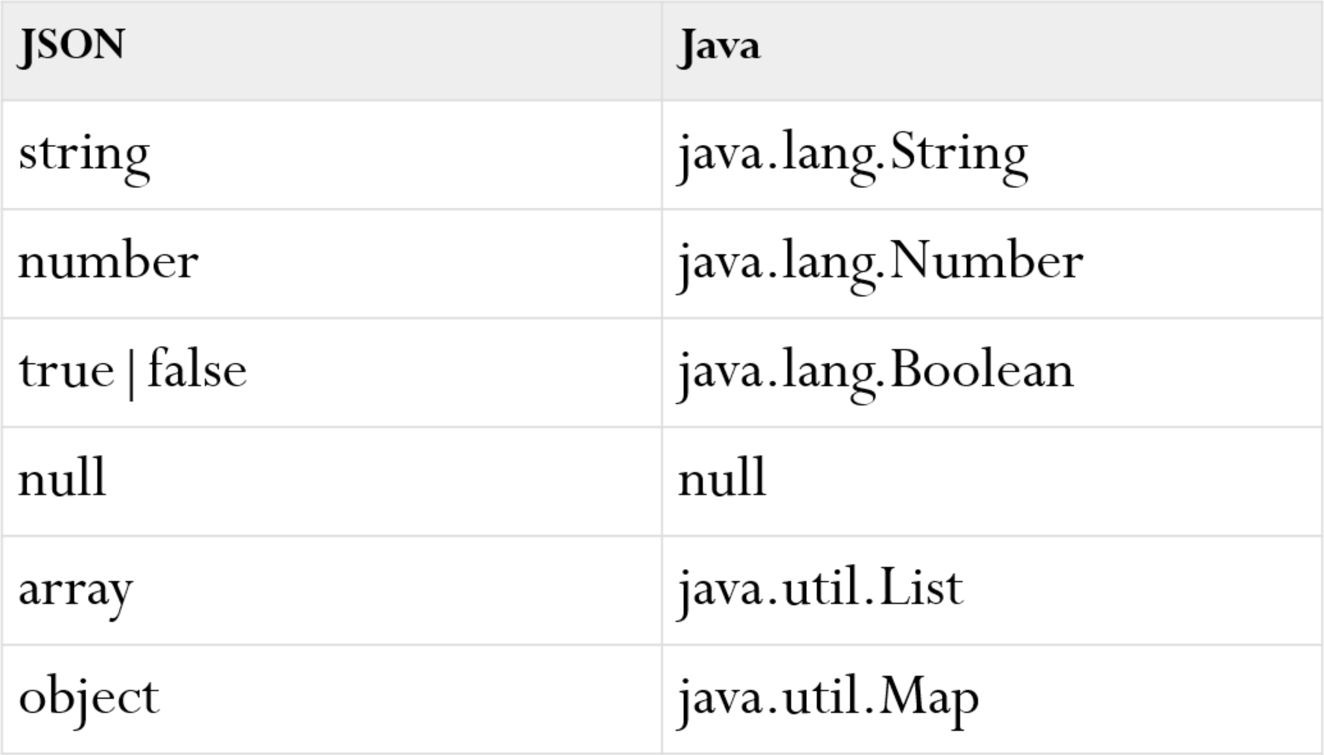
```
{
   "Book":
      {
         "Title": “Applying UML and patterns",
         "Authors": [ “Craig Larman" ],
         "Date": "2004",
         "Publisher": “Pearson"
      }
}
```

Processing JSON with Java

- Gson is a Java library that can be used to convert Java Objects into their JSON representation. It can also be used to convert a JSON string to an equivalent Java object.

  https://code.google.com/p/google-gson/

JSON and Java Entities



JSON	Java
string	java.lang.String
number	java.lang.Number
true | false	java.lang.Boolean
null	null
array	java.util.List
object	java.util.Map

Send/Receive JSON via Sockets

- Example: Send a student object request as a json to the server and receive a json response back.
- The code utilizes the google gson Java library to convert Java objects to json and vice versa.

Server Code

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

public class StudentServer
{
   public static void main(String args[]) throws IOException
   {
      final int PORT = 6789;
      System.out.println("Starting Server...");

      // create welcoming socket at port 6789
      ServerSocket welcomeSocket = new ServerSocket(PORT);
      while (true)
      {
         System.out.println("Waiting for a client...");

         // Wait, on welcoming socket for contact by client
         Socket connectionSocket = welcomeSocket.accept();

         // Start a thread with the client communication
         Thread clientThread =
             new Thread(new Communication(connectionSocket));
         clientThread.start();
      }
   }
}
```

Communication I

```
import java.net.Socket;
import com.google.gson.Gson;

public class Communication implements Runnable
{
   private DataInputStream inFromClient;
   private DataOutputStream outToClient;

   public Communication(Socket socket) throws IOException
   {
      // create input stream attached to the socket.
      inFromClient =
           new DataInputStream(socket.getInputStream());

      // create output stream attached to the socket.
      outToClient =
           new DataOutputStream(socket.getOutputStream());
   }
```

Communication II

```
   public void run()
   {
      try
      {
         // read line from client.
         String clientText = inFromClient.readUTF();
         System.out.println("Client> " + clientText);

         // convert from JSon
         Gson gson = new Gson();
         Student student = gson.fromJson(clientText, Student.class);

         System.out.println("Student: " + student);

         // creating reply
         Message reply = new Message(student, "Welcome");
         System.out.println("Reply: " + reply);

         // convert reply to Json
         gson = new Gson();
         String replyJson = gson.toJson(reply);

         // Send reply to client.
         System.out.println("Server> " + replyJson);
         outToClient.writeUTF(replyJson);
      }
      catch (Exception e)
      {
         e.printStackTrace();
      }
   }
}
```

Client Code I

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

import com.google.gson.Gson;

public class StudentClient
{
   public static void main(String args[])
         throws UnknownHostException, IOException
   {
      final int PORT = 6789;
      final String HOST = "localhost";

      // create input stream
      Scanner inFromUser = new Scanner(System.in);

      // create client socket, connect to server.
      Socket clientSocket = new Socket(HOST, PORT);

      // create input stream attached to the socket.
      DataInputStream inFromServer =
                      new DataInputStream(clientSocket.getInputStream());

      // create output stream attached to the socket.
      DataOutputStream outToServer =
                      new DataOutputStream(clientSocket.getOutputStream());
```

Client Code II

```
        System.out.print("Enter your name: ");
        String name = inFromUser.nextLine();
        System.out.print("Enter your student number: ");
        int number = inFromUser.nextInt();
        inFromUser.close();

        // create student object
        Student student = new Student(number, name);

        // convert to JSon
        Gson gson = new Gson();
        String json = gson.toJson(student);

        // Send line to server
        System.out.println("Client> " + json);
        outToServer.writeUTF(json);

        // Read line from Server.
        String serverReply = inFromServer.readUTF();
        System.out.println("Server> " + serverReply);

        gson = new Gson();
        Message reply = gson.fromJson(serverReply, Message.class);
        System.out.println("Message: " + reply);

        // Close connection.
        clientSocket.close();
    }
}
```