# Solution 1

# Centralized Java Chat

## 1 Overview

We present here the solution of Exercise 1. We first introduce the design and then explain the different classes we use. We also present the mechanisms used to connect/disconnect as well as for sending messages to the `ChatServer`.

## 2 Getting, Compiling and Running the Application

The complete source code is available for download at the exercise page of the course web site. The archive contains an ant (*http://ant.apache.org*) build file that can be used to compile the sources and generate the java documentation. To do this, simply type in the *Ex1* directory[1]:

```
ant
```

During the compilation, ant creates a jar file that can be used to launch the application:

```
java -jar dist/Ex1.jar
```

To clean the compiled classes, the javadoc as well as the jar file, simply type in a console:

```
ant clean
```

Finally, you can also compile and run the code using netbeans 5.0 (*http://www.netbeans.org*).

## 3 Design

The different classes are split in different packages: *client*, *serialization* and *server*. The design is the one presented in the statement of Exercise 1. However, we decided to make the `MyMessage` class be a subclass of the `Message` class. This allows us to rewrite only the `toString()` method.

### 3.1 Serialization package

This package contains the `IMessage` interface as well as the `Message` and `MyMessage` implementation. The `IMessage` represents the specification of a message that is sent between a client and a server. Its implementation contains a header and data. The header contains the username of the client that sends the message. The `MyMessage` class is a subclass of the `Message` one and simply redefines the `toString()` method that is used for printing a message.

### 3.2 Client package

This package contains two classes and one interface. The `IChatClient` represents the specification of a chat client and its implementation is `ChatClient`.

The `ChatClient` class contains a reference to a `IChatServer`. This reference is used in the `connect()`, `disconnect()` and `sendMessage()` methods.

[1]Please use the latest version of ant (1.6.5).

### 3.3 Server package

This package contains the `IServer` interface and its `ChatServer` implementation. Instances of `ChatClient` manipulate only `IServer`. As stated before, this makes the user independent of the implementation of a server and preserves the encapsulation of the `ChatServer`.

The `ChatServer` does not have any reference to `IChatClient` objects. Instead, the `ChatServer` contains a list of its connected clients (i.e., a `Vector` of `String` representing the usernames of the clients) and is responsible for managing the *connection*, *disconnection* of the clients as well as the *reception* of the messages.

### 3.4 CentralizedChat Class

This class belongs to the core package and is responsible for creating the different instances of `ChatClient` and the `ChatServer` as well as to call several methods on the instances of `ChatClient`.

## 4 Connection, Disconnection

When the `connect()` method of the `IChatClient` is called this, in turn, calls the `connect()` method of the `IServer`. When the `ChatServer` receives this method call, it puts the new client into its list of clients (i.e., it puts the username into its `Vector` of usernames).

A call on the `disconnect()` method of the `IChatClient` calls the `disconnect()` method on the `IServer`. The `ChatServer` removes the client from its list of clients (i.e., removes the username from its `Vector` of usernames).

## 5 Sending, Receiving Messages

When the `sendMessage()` method of the `IChatClient` instance is called, the `ChatClient` calls the `sendMessage()` method of the `IServer`. Finally, when the `ChatServer` receives the message, it simply prints it on the standard output stream.