Purpose	Learn to apply a framework in developing an application
Files	Download OCSF 2.31 zip file ocsf-231.zip. The file includes JAR file, Javadoc, and source code. Unzip to install on your computer, but **not** in your Java project directory. Its OK (but not necessary) to copy the ocsf-2.31.jar to your project directory. Add **`ocsf-2.31.jar`** to your project.
What to Submit	Create a project named **`ocsf-lab`** on Github and commit your solution.

Overview

The purpose of this lab is to practice using a framework. OCSF is a simple framework that provides TCP-based client-server connections.

Without the framework, we would need several labs just to implement simple network communication.

You will see that a framework reduces development time and usually results in higher quality software. Most frameworks are extensively tested both by developers and users, so they typically have fewer bugs and a better architecture than software written from scratch.

Using OCSF

The OCSF framework is described in the book by Lethbridge (chapter on OCSF is on class web).

Your client uses methods of OCSF's AbstractClient or ObservableClient classes to connect to a server and send/receive objects. ObservableClient extends java.util.Observable; it is useful for writing a client as an *Observer*. That way the client is *notified* (updated) when a message is received from the server.

The first step is to create a subclass of AbstractClient or ObservableClient.

Callback Methods (Don't Call Us, We'll Call You)

The framework has several *callback methods* that you can ***override*** in your client class (a subclass of AbstractClient or ObvervableClient). The framework calls these methods when an event occurs. These methods are how you use the framework to provide functionality to your app.

`handleMessageFromServer`	(**required**) this method is invoked whenever the client receives a message from the server
`connectionClosed( )`	(optional) this method is invoked if the connection to server is closed.
`connectionEstablished()`	(optional) this method is invoked when a connection to the server is established.

For other callback methods, see the handout and OCSF Javadoc.

Control and Utility Methods

These are methods that you *invoke* to tell the framework to do something or perform a query. They are provided by AbstractClient. Don't override these methods (unless you have a genuine reason to do so, and probably call the superclass method as part of your override).

`sendToServer( Object )`	send a message to server. May throw exception.
`openConnection( )`	attempt to connect to server. May throw exception.
`closeConnection( )`	close the connection.
`isConnected()`	test returns true if currently connected to a server

Problem 1: Write a Client that connects to server and sends Strings

You can write a console-based client or GUI client (using Swing).

Create a client as a subclass of AbstractClient or ObservableClient. that does this:

a) Connect to the server and display a "connected" message.

b) Print (or display in GUI) any messages from the server.

c) Accept input from the user at console (or in GUI) and send it to the server.

d) Receiving messages and sending messages are asynchronous operations. Your code for accepting input from the user should **not** be in the "handleMessageFromServer" method.

e) Close the connection when you signal that you want to quit. If you write a GUI, there should be a "Disconnect" button. On the console, you can decide what input means "disconnect".

Example:

```
Connected to server 158.108.32.99    <-- message from your program
> Hello. Please Login                <-- message received from server
Input:  Login Fataijon               <-- send "Login yourname"
> Hello Fataijon.  What is 8 & 12         (Prove you are a programmer)
Input:  8
> Correct!   What is 8 ^ 12                (These are bitwise operations)
Input:  12
> Sorry, wrong answer.
Input: 4
> Correct!
Input: quit
Disconnected.
```

1.1 Write a client class that extends AbstractClient or ObservableClient. ObservableClient provides the same methods as AbstractClient and also extends Java's Observable. This is useful for receiving notification (in a separate thread) when a message is received from the server.

1.2 Override the handleMessageFromServer(Object message) method. Print or display the message.

1.3 Use the constructor to set the address and port of the server to connect to.

1.4 Write a run() method the connects to server by calling openConnection().

1.5 Use the other "control methods" (see first page) to send messages and close the connection.

1.6 Override the other "callback methods" and tell the user when he is connected to server or disconnected from server.

When you complete this task, your name will be added to the scoreboard along with your score.

See end of this handout for UML diagram of AbstractClient.

See the Javadoc (in ZIP file) for details of using the methods.

Problem 2: Write a Chat Server for 1-to-1 Chat

Write your own server using OCSF's AbstractServer class.

You should create a server that requires clients to identify themselves, so you know which user is connected on which ClientConnection object.

2.1 Write a class that extends AbstractServer or ObservableServer (both classes have the same methods).

2.2 When a new client connects, you should wait for the client to (somehow) identify the user. Design your own solution to this. The ClientConnection object has a map that you can use to store arbitrary values. You can use this to store the user's name. For example:

```
client.setInfo("username", clientname );
```

2.3 When a user logs in, servers send a message to all clients telling them "*Clientname* connected" (you can design the format of this message. It doesn't have to be a String.)

2.4 When a logged-in client sends a one-to-one message like this:

```
To:  Anchan
Hi, Anchan. How are you?
```

your server should find a client connection with login name "*Anchan*" and send the message. Be sure to tell Anchan who the message is from!

2.3 If a client sends the String message "Logout" then close the client connection and tell all other clients "*Anchan logged off*".

2.4 If the client sends any other message, the server responds that message is not recognized.

How to Record the Client Name

OCSF creates a ClientConnection object for each connected client, and passes this object as a parameter in handleMessageFromClient.

You can save the client name using the setInfo / getInfo methods. For example:

```
// setInfo is a HashMap. You can use can String as a key
client.setInfo( "username", loginname );
```

How to Find a Client By Name

OCSF AbstractServer doesn't provide an easy way to search the current ClientConnection objects, so you can find a client by username.

One solution is to maintain your own list of **`ClientConnection`** in your server class.
Override the *callback* methods:

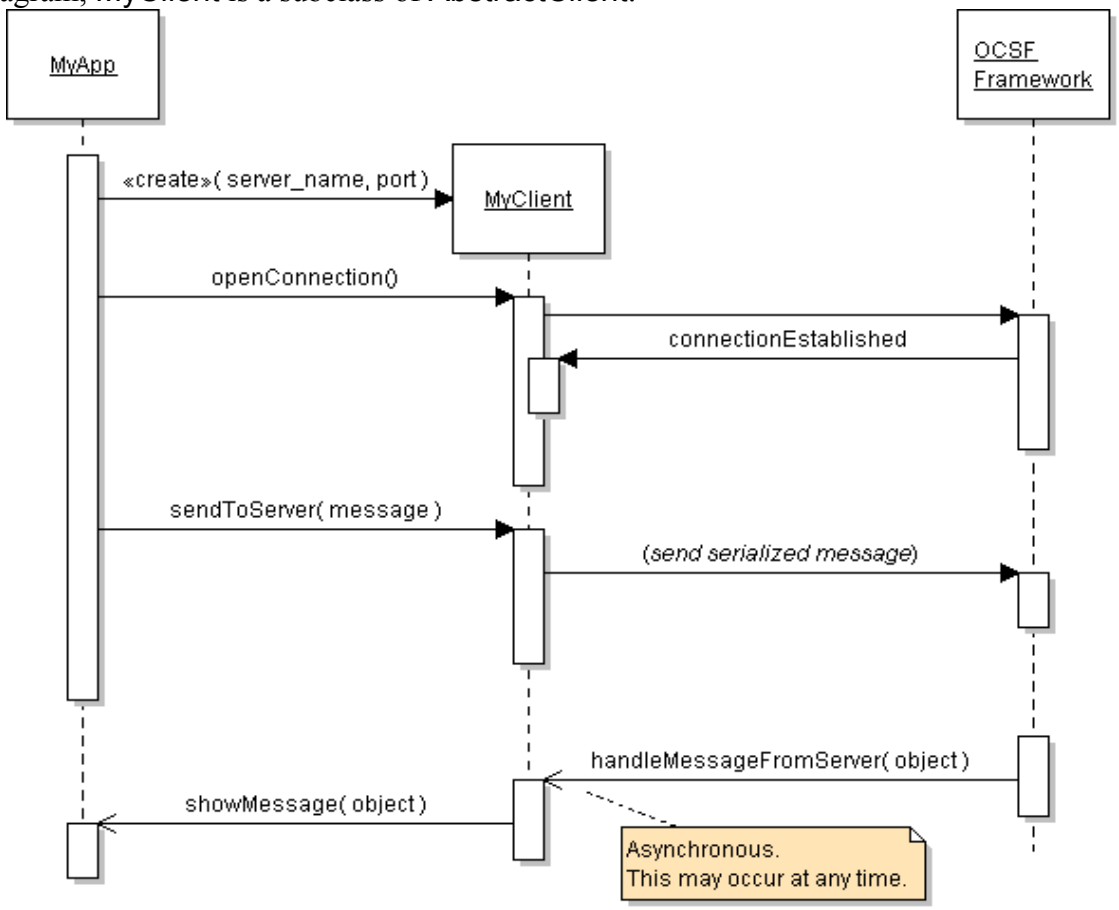
`clientConnected( ClientConnection conn )` - a new client is connected

`clientDisconnected( ClientConnection conn )` - a client has disconnected

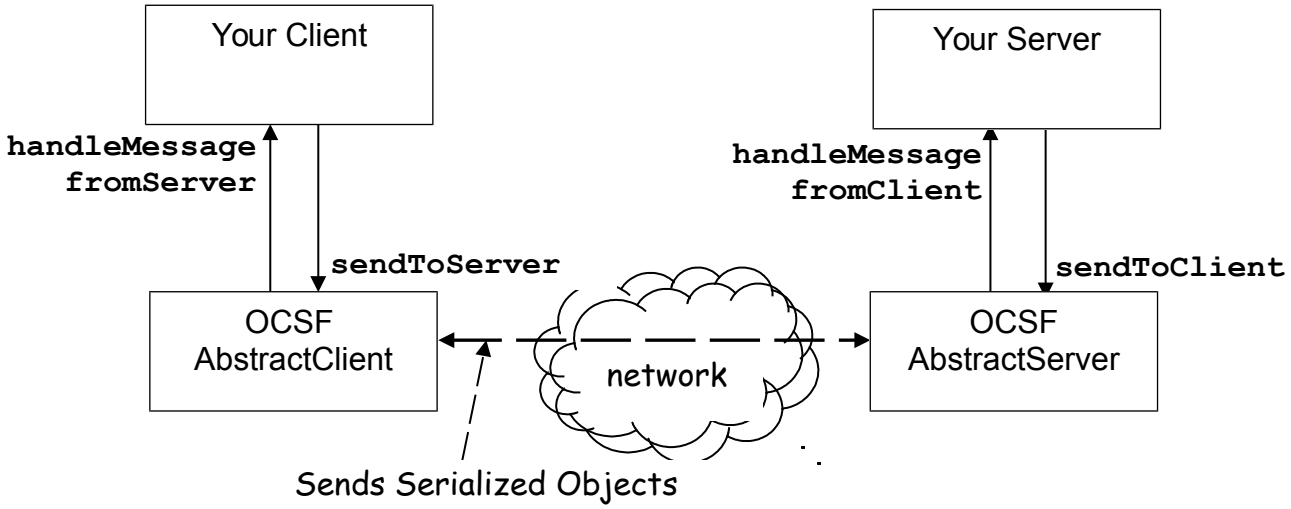
use these hooks to add client connection to your List and remove disconnected clients from your List.

Typical Usage

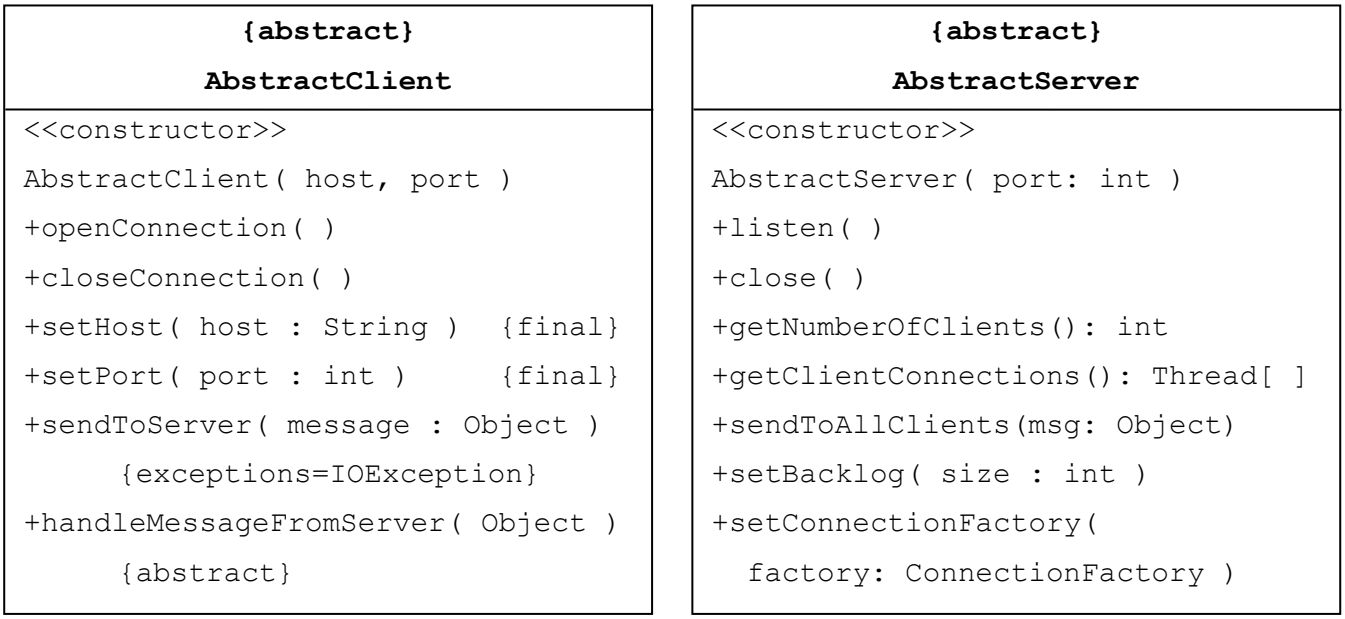
In this diagram, MyClient is a subclass of AbstractClient.



Conceptual View of OCSF Operation



UML



References

Lethbridge and Lagariere, *Object-Oriented Software Engineering,* 2E. Textbook describes use of OCSF and a chat project.

A standard, high-performance framework for chat and other applications is XMPP.

XMPP is a standard protocol for real-time messaging; XMPP was originally called *Jabber*. Google Talk uses XMPP. You can use XMPP to write your own Chat client or other Internet application. There are many several free XMPP servers (such as *Jabberd* and *OpenFire*), clients, and libraries. XMPP can be used for more than just chat.

SMACK is an open-source XMPP library for Java. It is used by several chat applications. http://www.igniterealtime.org/projects/smack/

- How to use SMACK to write a Java client: http://www.javacodegeeks.com/2010/09/xmpp-im-with-smack-for-java.html
- Other two articles in the same series describe infrastructure for using XMPP.

XEP-0045 Multi-User Chat. Protocol for a multi-user chat using XMPP. http://xmpp.org/extensions/xep-0045.html#bizrules-message