# High Level Architecture Description

From the external point of view, the **Virtual Class Room** system consists of two main parts:

**Virtual Class Server** and **Virtual Class Client**.

**Virtual Class Server** manages message passing between clients. This is service application that should run in the background.

**Virtual Class Client** provides ability for all client types (both Teacher and Student) to send and receive messages. This is GUI application.

From the internal point of view, the Virtual Class Room systems consists of the following Java packages:

* ***port\_channel*** - contains network communication core classes, **ChannelPort** and **ChannelEndpoint**, **Message** class and other supplementary classes around them.
* ***virtualclassroom.protocol*** - contains Virtual Class Room specific protocol parts
* ***virtualclassroom.client*** - contains class **ClassroomClient**, which handles communication with Virtual Class Room Server
* ***virtualclassroom.clientgui*** - here resides the client GUI program
* ***virtualclassroom.server*** - here resides the server program
* ***com.eclipse.wb.swing*** - supplementary classes automatically generated by the WindowBuilder Forms Designer - required for GUI part

# Server Part

Overall schema of how server works is as follows:

* Reads configuration file or applies default settings for various parameters
* Starts separate thread that listens to server socket and all accepted connection using Java NIO channels interface
* Picks up messages generated by the network communication thread and processes them, including sending and broadcasting messages to various clients

# Client Part

Overall schema of how GUI client part works is as follows:

* End-user launches the client GUI program
* End-user enters server location and other required data and connects to server
* End-user can be either a Teacher or Student , everyone use the same GUI, but teacher needs t specify additional Teacher Password in order to be identified as teacher.
* After connection succeeds, end-user can enter messages in the text area, and send them using the **Send** button.
* End-user can see all incoming messages and supplementary notifications in the top are of Client GUI window.
* End-use finally disconnects from server or closes GUI.

# Details About Networking

## Networking Overview

Server and client use TCP/IP transport protocol to communicate between them, but use different networking APIs, which best fit for the concrete situation.

## Server-Side Networking

Server uses Java NIO Channels API for the network communication, which is more suitable for the cases when multiple network connections must be handled simultaneously.

First, server sets up ***ServerSocketChannel*** and registers it within the ***Selector*** for ***ACCEPT*** operation.

When ***ACCEPT*** operation happens, server creates new ***SocketChannel*** and registers it within the ***Selector*** for the READ operations.

When ***READ*** operation happens, server first attempts to send heartbeat message to that channel. This way dead connections are detected, because sending will fail on them. After heartbeat sent successfully, server attempts to read message from the selected ***SocketChannel***. Any network read error or data format error results socket channel being closed.

When message is read successfully, it is put into the message queue, from where it can be picked by a message consumer thread.

***Broadcasting*** messages is implemented as manual sending the same message to the each connected client. I/O exception in the middle of broadcasting client chain is saved, and thrown in the end as MultiException, containing all exceptions happened during broadcast operation on the all clients.

## Client-Side Networking

Client uses traditional Java network socket for communication with server, which best fit for the situation when there is single network connection required.

Client starts separate network data receiver thread, which first establishes the connection with Server and sends ***HELLO*** message to a server. ***HELLO*** message contains end-user's nick name and optionally, teacher's password. For simplicity of protocol, password is not encrypted or protected in the any other means, just clear text, it is sufficient for the demo purposes, however, in the real production systems, encryption and hashing must be used.

After that, as long as end-user enters some text, typically through ***CHAT*** or ***ADMIN*** messages, and presses Send button, text is converted to message of the type ***CHAT*** and transmitted to server.

Meanwhile, when some data is received in the receiver thread, it is passed to a callback which published data to the visual data view (based on the visual component JTable), and adjust connection state - i.e. may close connection or display connection details in the status bar.

Any network communication failure or data format inconsistency cause connection to server to be closed. Client can connect again after that as usually. Communication failures are represented as special type of messages, ***COMMUNICATION\_ERROR*** and ***CLIENT\_DISCONNECTED*** which are also being passed to the callback.