USER DOCUMENTATION – A1

[A1-Bd]

Contents

1 Introduction

This document provides a brief overview on how to use the package `no.ntnu.fp.net.co`, also know as A1. The first section will describe the interface between A1 and the application. The next section will describe some of the code distributed in A1.

2 Interface description

The connection-oriented interface is defined by the following primitives in the interface `Connection`:

- `accept()` – Listen for incoming connections, and return a Connection instance representing that connection.
- `connect(InetAddress remoteAddress, int remotePort)` – Establish a connection to a remote host at `remoteAddress` that is listening for incoming connections on `remotePort`. When this method returns, the instance on which it was called is expected to represent the newly created connection, if no exception was thrown during execution of the method. The address `remoteAddress` can be retrieved by using the static method `java.net.InetAddress.getByName("localhost")` defined in the Java standard library.
- `send(String msg)` – Send the message msg to the remote host. This method throws an exception if a connection is not established or the message was not successfully delivered.
- `receive()` - Wait for a message to arrive from the remote host. This method throws an exception if a connection is not established when the method is called, or if it received a disconnect request.

- `close()` – Close the connection to the remote host. When this method returns, it is not possible to send or receive before the connection is re-established using accept() or connect().

3 Textual description

This section explains how to use the classes and interfaces constituting the interface of the connection-oriented communication package to the application.

3.1 Using the Connection-interface

A server application will typically create an instance of a class implementing `Connection` and repeatedly call `accept()` on this object to accept new connections. The new connections are most conveniently handled in threads of their own. On the client side, a single Connection is employed for each server the application wants to connect to, and the connection is created using connect() on that object.

When a connection has been established, messages are sent across the connection using the send() and receive() primitives. Note that receive() must have been called on the receiving side prior to attempting to send(), i.e. no queuing of packets is done. Moreover, the connection is half-duplex, meaning that only one side may send at a given time.

The connection is closed by calling close() on the connection, on either side. When this primitive is called, an exception is raised on the opposite side stating that a disconnect has been requested. The appropriate action on the server side is then to call close() on the connection instance, signalling that the application has been made aware of the disconnection.

The interface is modeled after the Socket class in the Java standard API, and reading the socket tutorial[1] serves as an explanation of how to use the interface in an application.

3.2 Helper classes

`ReceiveWorker` can be used to simplity asynchronous handling of incoming messages. Application objects who wants to receive messages, implement the interface `MessageListener`, and register themselves to this class. While the parameter `running` is `true`, the `ReceiveWorker` will listen for incoming messages by calling `receive()` on the associated connection. When this method returns a message, it is given to all the objects who have registered as a message listener.

It is, of course, not mandatory to use these helper classes/interfaces.

1 http://java.sun.com/docs/books/tutorial/networking/sockets/