DOCUMENTATION OF THE DISTRIBUTED CODE – A1

[A1-Doc]

Contents

1 Introduction

Some of the code for the class `ConnectionImpl` is provided to relieve the required amount of programming to realise the interface `Connection` (see [A1-Ud]). This document explains the code that has been released. It is important to emphasise that even though code is provided, it is not required that this code is used; the code is to be regarded as a mere suggestion and a help to getting started. Knowledge of programming with threads is essential in this project. Please consult a book on Java programming or the Java Threading Tutorial[1] for more on threads and handling of such in the Java programming environment. Reading the user documentation of A1 (see [A1-Ud]) and A2 (see [A2-Ud]) should also be helpful.

2 High-level description

This section briefly introduces the distributed code. Several classes are included:

- `AbstractConnection` – Base class for ConnectionImpl, implementing parts of the functionality.
- `Connection` – The interface you are required to implement, and that the application relies on for connectivity.
- `InternalReceiver` – Receives packets in a separate thread.
- `SendTimer` – Sends packets in a separate thread.
- `SimpleConnection` – Connection realisation that circumvents A2 for testing purposes.
- `ConnectionImpl` – Starting point for implementation. This is the only class that you should need to change, filling in method bodies.

1 http://java.sun.com/docs/books/tutorial/essential/threads/

3 Main classes: The Connection implementation

The two classes you need to be concerned with are: `AbstractConnection` and `ConnectionImpl`. Utility routines implemented in AbstractConnection are intended to facilitate the realisation of the Connection interface. Specifically, these methods include (see source code and javadoc for a more verbose explanation):

- `constructInternalPacket(flag)` – Construct a packet with the given flag (ACK, FIN, etc.. Use this to create the internal packets (i.e. flagged packets) you want to send.
- `constructDataPacket(payload)` - Construct a packet with no flag and the give payload. Use this to create the data packets you want to send.
- `receivePacket(internal)` – Receive a packet from the other side. Use this when receiving, instead of using A2 directly, because this method helps keeping the protocol transparent to the application by holding back protocol packets when called from the application and holding back data packets when protocol packets are desired. An EOFException might be thrown from this method if a FIN-packet is received. If such an exception is thrown, the disconnect request is stored in the variable 'disconnectRequest'.
- `receiveAck()` – Wait for a packet with ACK or SYN_ACK flag.
- `sendAck(packet, boolean)` – Construct and send ACK for the given packet, using SYN_ACK if the boolean is true, ACK otherwise.
- `simplySendPacket(packet)` – Sends a packet, abstracts away A2 but does not do much else useful.
- `sendDataPacketWithRetransmit(packet)` - Sends a data packet and waits for an ACK, will retry if no ACK is received.

You are supposed to complete this by implementing the following methods in `ConnectionImpl`:

- `accept()` – Wait for an incoming connection, perform the server-side part of a three-way handshake and return a new instance initialised with values appropriate for the new connection that has been set up. Tip: You need to negotiate new port numbers for the new connection.
- `connect(address, port)` - Connect to a remote instance using a client-side three-way handshake.
- `close()` – Close the connection using a four-way disconnection procedure. Note that this handles both active and passive disconnect.
- `send(message)` – Send the given message (`String`) to the other side, making sure it really arrived. If the message could not be sent (after a number of attempts), throw an exception to indicate that the link is faulty.
- `receive()` – Wait for a message from the other side and return it.
- `isValid(packet)` – If you subclass `AbstractConnection`, you need to fill inn this method, checking the given packet for errors and returning true if the packet has not been damaged, false otherwise.

4 Auxilliary classes

The other classes should not need to be altered. See the source code of `AbstractConnection` for an illustration of how to use these classes if you need them.

5 References

This document references to the following documents:

[A1-Ud] User documentation – A1
[A2-Ud] User documentation – A2
[A1-K] Code – A1