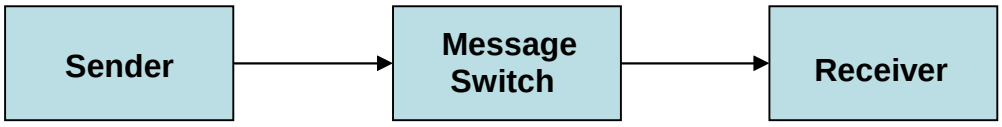
PROGRAMMING ASSIGNMENT 3

Message Switch Application

Initial Stage

In this relatively simple programming exercise you are going to design, implement and test a message switch application as explained below. The architecture of the communicating programs is shown on the following figure.



The MessageSwitch program should be started first, waiting for the receiver to connect to first. After the Receiver connects to it, the sender can also connect and start sending messages which should be delivered to the receiver. A running Sender should listen for messages typed by the user through the keyboard and should forward those messages to the Receiver through the MessageSwitch. The Receiver should print out the messages received to the screen. The communication between those running programs should be based on TCP (i.e. stream) sockets. Apart from the data messages that these processes exchange, there should also be a special message that notifies the Receiver that a Sender has finished its sending session, as described in detail below.

- **Sender:** this process should accept a "message" typed by the user on the keyboard, forward it to the MessageSwitch and then wait for another message. A message being typed on the keyboard is considered complete when the user, having typed a series of characters, hits the ENTER key. When the user wants to terminate the session, s/he should send a special message that consists of the single word "quit" (with character case not being significant). As soon as this message is typed by the user, the Sender should send it to the MessageSwitch and then terminate gracefully, releasing any streams and sockets it has been using.
- **MessageSwitch:** when this process has a connection with the receiver and the sender, it should be reading the messages received by the Sender on the relevant TCP socket and forward them to the Receiver using the TCP socket connecting to the latter. When the message switch receives the termination message from the Sender, i.e. "quit", it should also relay it to the Receiver and terminate gracefully.
- **Receiver:** this process should receive the messages sent by the MessageSwitch and print them out on the screen. It should terminate gracefully when receiving the termination message from the Sender viva the MessageSwitch.

These processes should in principle be able to run on different machines on the network although they could also be run on the same machine for simplicity. As such, the Sender and Receiver should accept a command line argument of the machine name where the MessageSwitch runs. If this argument is not given, the program logic should assume that the MessageSwitch runs on the local machine and use the "loopback" IP address 127.0.0.1 to connect to it. You may assume that the Sender and Receiver know beforehand the TCP port on which the MessageSwitch listens, so this number may be "hardwired" in their logic. On the other hand, if you want to make the application more flexible, you could pass this port number as a second argument to the Sender and Receiver, e.g. "*java Sender <machine_name> <port_no>*". The port number used should be bigger than 1024 as ports up to that number are

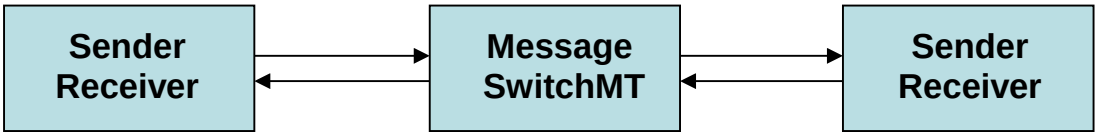
reserved for well-known processes. You should also make sure you close every socket you use in your program when the latter terminates. If you terminate a program not cleanly and the socket remains open, next time you are going to use the same socket which you just used but did not close, you may get errors.

You should implement these programs so that they print out information messages when something happens e.g. when a program connects successfully to another program, when it sends/ receives a message and when it terminates.

The starting point for doing this exercise should be the EchoClient and EchoServer programs as in the notes, so you should implement and experiment with them first.

Second Stage

In this second part of this programming exercise, you are going to modify the Sender to make it a **SenderReceiver** so that it can also receive messages and you will also modify the MessageSwitch to **MessageSwitchMT** (multi-threaded) so that it can now handle bidirectional communication. A "quit" message from either of the SenderReceiver programs should result in terminating all the running programs. The architecture of the communicating programs is shown on the following figure.



Because reading input from a BufferedReader object with the readLine method results in a blocking call until input appears, both the SenderReceiver and the MessageSwitchMT will need to be multithreaded so that they can get input simultaneously from two streams: the SenderReceiver programs from the standard input and the socket to MessageSwitchMT and the MessageSwitchMT from the sockets to the two SenderReceiver programs.

In the SenderReceiver program, one of these blocking reads can take place in the "main" thread, e.g. the one listening for input from the keyboard in a *run* method, in a similar fashion to the EchoClient program in the notes. The other should be a separate thread instantiated and started just before the run method is called in the main program method. In the MessageSwitch2, the two threads listening for and relaying input are symmetric as they simply copy input from one socket to the other. As such, they can be implemented through exactly the same thread whose constructor takes arguments in the right order for each of them. As such, after some initialisation the two threads could be instantiated and started.

A key aspect of this programming exercise is *incremental development* and *prototyping*. You could implement this application in a gradual fashion as follows.

1. You should first design and implement a simple MessageSwitch that waits for a single Sender to connect to it and then prints out the messages it receives.

2. You should then extend the MessageSwitch so that it first waits for the Receiver to connect to it, then for the single Sender and subsequently relays messages from the Sender to the Receiver.

3. You should then extend the MessageSwitch to the multi-threaded MessageSwitchMT in order to handle bidirectional input while the SenderReceivers are still single-threaded with predefined order of operation: SenderReceiver 1 sends something and then waits for a

response before being able to send again while SenderReceiver 2 first waits for something and then allows its user to send. You should use a single program for the SenderReceiver in both these roles.

4. Finally, you should also make the SenderReceiver multithreaded so that there is no order in the conversation and they could both send data at any time to each other.

For dealing with input and output, you should use the Java classes BufferedReader and PrintWriter for which you can find documentation regarding their use in the lecture notes and on the Web.

Deliverables. You should produce a relatively small document (4 pages maximum) describing the relevant issues and program design for these programs and append the Java code files in 2 pages per page format. Java code should only be provided for the second stage programs – you do not need to include the simpler previous versions unless of course you did not manage to implement the Stage 2 programs in which case you should of course include the ones of Stage 1.

An examination through demonstration of each candidate will take place on the 28 April 2-6pm in term 3, just after the Easter break; the examination also will be the time you should be handing in the above deliverable.