HangmanServer.java contains the logic of the game.

A guessWord is used to track the current status of guessed list. Each time if a correct guess is made a "-" in guessWord is replaced by correct guess. "life" is to track players life. Each time a guess is made BroadCastMessage() is called to write the message to players.

The game uses two functions **BroadCastMessage()** and **ClearHandlers()** that are implemented in dispatcher. The purpose of first is to report players of every moves i.e. updating status of the game. A Boolean variable **gameOver** is set when game is over that calls the later function. This, when game is over, closes and removes all the handles.

AcceptHandler.java handles all the events i.e. socket connections and later on string inputs. **TCPTextHanlder,** class within AcceptHandler class, does the tcptext handling i.e. reading/writing text from the sockets created earlier in accepthandler.

In **Dispatcher,** handlers and events lists are created. Function handleEvents() handles the events if there is any.

Function select() takes the first event from the queue and processes it.

Function addHandler() adds the new handlers. For each handler a new thread is started and then these handlers handle their events simultaneously. To start the listening of new events, thread is implemented using anonymous threading technique in java.

Read() waits until an event is received on the handler and when an event is received insert() function is called where event is listed in the corresponding handler.

Function removeHandler() removes the handlers from the dispatcher lists and so now no corresponding handles are processed.

Function BroadCastMessage() is used to write the game status to each players' console by using their corresponding handlers.

Function ClearHandlers() is used to close all tcptexthandle and accepthandle and removes all the events. This function is called when the game is over.

We have implemented the hangman logic using Keep It Simple logic. And the test also shows that the logic is fine. Throughout the code, we have used the concept of inner classes and statement reduction wherever possible. It is quite interesting to notice that the reactor pattern itself manages the concurrency problem, that is we don't have to explicitly define and use the concurrency scope and avoidance technique. However, we have employed the synchronized construct of java.

Problems

At one place in dispatcher, in insert (), when we were trying without giving any limit then our overflow test was hung up.

Then we followed the newsgroup "http://news.tky.fi/article.php?id=1729&group=opinnot.tik.rinnakkaisohjelmointi#1729" and following that we limited the events number upto 150 and it worked.

Testing

We tested the code with given test2.jar. It shows 11 tests ok. We couldn't pass the stress tests.

Using telnet in command line we played from 5-6 consoles and it worked fine. We were able to see each other's guessed word, win the game or lose the game. Game was running fine.

Submitted By:

Fariha Nazmul(84999P)

Sameer K.C. (84967W)