3/26/2018

Chat Server Programming Assignment

Vinod Thorat

VINOD THORAT

Chat Application

Problem Statement

Chat Server Programming Assignment
==================================
Introduction

You have to write a program based on provided requirements. You are free to use a programming language of your choice, preferably Java. Please state all assumptions when implementing a solution. It is important to deliver elegant object-oriented code, apply design patterns and consider performance of the implementation. In addition, please provide JUnit test cases and comments for the implemented classes. Your delivered program should compile (provide all required jars) and run.

Chat Server

Develop a TCP/IP based multi-party chat server and client. Application should include appropriate unit test, error logging, and design documentation.

Server Requirements:
- Server should accept 1 to 10 simultaneous client connections
- A message received from any client should be echoed to all clients by default
- The server should log all client connect and disconnect events
- The listening IP and port address should be configurable

Client Requirements:
- The server IP and port address should be configurable
- The client should attempt to connect to the server and gracefully handle connection failures and disconnect events
- At start-up, the client should ask the user to provide a unique username
- The client should allow the user to enter an alpha-numeric text message
- Entered text messages should be transmitted to the server
- The client should display received text messages
- The client should be able to filter messages by user
- (Optional) The client should be able to send a message to a selected user

Chat Application

Solution

The solution for given assignment is in two phases.

1. Screenshots of all over application.
2. Source code for given application.
3. Junit testing.
4. Logging file.
5. Two applications data.

Source code

Junit Testing Code:

ClientCodeTest.java

```
package chat_server;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.util.Date;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class ClientCodeTest {
    /**
     * Set up logic for client information.
     * @throws Exception Thrown in case of any error.
     */
    @Before
    public void beforeTest() throws Exception {
        System.out.println("Testig : Preparing for test execution: " +
getClass().getSimpleName());
    }

    /**
     * Shut down logic for given port socket.
     * @throws Exception Thrown in case of any error.
     */
    @After
    public void afterTest() throws Exception {
        System.out.println("Testig : Shutting down test execution: " +
getClass().getSimpleName());
    }

    @Test
    public void testClientCode() {
        try {
             System.out.println(getClass().getSimpleName() +" Wait 6
seconds server to start and get connection");
             Thread.sleep(6000);
             final String hostName = "localhost";
             Socket socket = new Socket(hostName, 2222);
             ObjectOutputStream ou = new
ObjectOutputStream(socket.getOutputStream());
             ObjectInputStream in = new
ObjectInputStream(socket.getInputStream());
             ou.writeObject("THIS IS TESTING FOR CLIENT COMMUNICATION
>>>>>>>>>>>>>>>>>>> \n\n\n");
             ou.flush();
             ou.close();
             in.close();
           } catch(Exception e) {
             System.out.println(e.getMessage());
          }
    }
}
```

```
ServerCodeTest.java

package chat_server;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.Date;
import org.junit.After;
```

Chat Application

```
import org.junit.Before;
import org.junit.Test;

public class ServerCodeTest {
    /**
     * Set up logic.
     * @throws Exception Thrown in case of any error.
     */
    @Before
    public void beforeTestExec() throws Exception {
        System.out.println("Testing : Preparing for testcase 
execution: " + getClass().getSimpleName());
    }

    /**
     * Shut down logic.
     * @throws Exception Thrown in case of any error.
     */
    @After
    public void afterTestExec() throws Exception {
        System.out.println("Shuttig down test execution: " + 
getClass().getSimpleName());
    }

    /**
     * Example test ServerCode method.
     */
    @Test
    public void testServerCode() {
      boolean isCheck = false;
      ServerSocket serverSocket = null;
      final String hostName = "localhost";
      try {
          InetAddress inetAddress = InetAddress.getLocalHost();

          if (hostName.compareTo(inetAddress.getHostName())==0){
              serverSocket = new ServerSocket(2222);
              while (!isCheck){
                  Socket clientData =  serverSocket.accept();
                  System.out.println(getClass().getSimpleName()
+"Getting Data " + clientData.getInetAddress());
                  ObjectInputStream objectInputStream = new 
ObjectInputStream(clientData.getInputStream());
```

```
                  ObjectOutputStream objectOutputStream = new
ObjectOutputStream(clientData.getOutputStream());
                  final Date checkdatewithObject  =
(Date)objectInputStream.readObject();
                  System.out.println(getClass().getSimpleName() +"
read data object from client " + checkdatewithObject);
                  objectInputStream.close();
                  objectOutputStream.close();
                  isCheck = true;
              }
          }
      } catch (UnknownHostException e) {

          e.printStackTrace();
      } catch (IOException e) {
          e.printStackTrace();
      } catch (ClassNotFoundException e) {
          e.printStackTrace();
      }finally{
          try {
              if(serverSocket != null){
                  serverSocket.close();
              }
          } catch (IOException e) {
              e.printStackTrace();
          }
      }
      System.out.println(getClass().getSimpleName() + "Output from
test class and method for connecting.");
    }
}
```

Test_Client_Server.java

```
package chat_server;

import static org.junit.jupiter.api.Assertions.*;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.ServerSocket;
```

Chat Application

```
import java.net.Socket;
import java.util.concurrent.Semaphore;

import org.junit.Before;
import org.junit.jupiter.api.Test;


class Test_Client_Server {

	 private ServerSocket serverSocket;
	 private OutputStream serverOut;
    private InputStream serverIn;
    /**
     * Shared lock between the "client" and "server" code, to make the
test case
     * synchronous.
     */
    private Semaphore checkLock = new Semaphore(0);


	 @Before
    public void before() {
		System.out.println("@Before");
		Thread myThread = new Thread() {
			public void run() {
				System.out.println("@Before myThread
run()");
				try {
					ServerSocket myServer = new
ServerSocket(2222);
					System.out
							.println("@Before
myThread run() - server socket created.");
					myServer.accept();
					System.out
							.println("@Before
myThread run() - accepted connection");
				} catch (IOException e) {
					e.printStackTrace();
				}
			}
		};
		myThread.start();
    }
    @Test
    public void itDoesntAssertInSeparateThreads() {
      Thread otherThread = new Thread(new Runnable() {
        @Override
        public void run() {
          assertTrue(false);
        }
      });

      otherThread.start();
      assertTrue(true);
    }




   /**
    * Tests server and client side sockets in one flow. A lock object
is used for
    * synchronous between the two sides.
    */
   @Test
   public void testClientServer() throws IOException,
InterruptedException {
      ServerSocket server = new ServerSocket(2020);

    // ServerStart sc = new server_frame.ServerStart();
      listen(server);

      Socket client = new Socket("localhost", 2020);
      client.getOutputStream();
      client.getInputStream();

      System.out.println("Waiting for lock");
      checkLock.acquire();
      System.out.println("Acquired lock");


      client.close();
      server.close();
   }
```

```
    private void listen(ServerSocket server) {
        new Thread(() -> {
          try {
            Socket socket = server.accept();
            System.out.println("Incoming connection: " + socket);

            serverOut = socket.getOutputStream();
            serverIn = socket.getInputStream();

            checkLock.release();
            System.out.println("Released lock");
          } catch (IOException e) {
            e.printStackTrace();
          }
        }).start();
      }


}
```

This is JUnit code for client and server, uploaded at GitHub account @vinodthorat121
Under ChatServerApplication.

Thank you,
Vinod Thorat