Programmieren 2
Selbststudium Semesterwoche 10

Aufgabenstellung
Implementieren Sie eine Chat Anwendung zwischen zwei Teilnehmern /Teilnehmerinnen gemäss folgender Spezifikation. Ergänzen Sie hierzu den Rahmen ChatRahmen.zip, der sich auf ILIAS befindet.
Ein beispielhafter Chat könnte folgendermassen aussehen:

```
C:\WINDOWS\system32\cmd.exe

D:\Unterricht\Programmieren II\SS2007\SW 10\ChatRahmen>java Chat
Input 2: Hallo
1 receives: Hallo
Input 1: Hallo, wie geht es Dir?
2 receives: Hallo, wie geht es Dir?
Input 2: gut, was machst Du heute Abend?
1 receives: gut, was machst Du heute Abend?
Input 1: weiss noch nicht. Hast Du Lust auf Kino?
2 receives: weiss noch nicht. Hast Du Lust auf Kino?
Input 2: Ja,... ich ruf Dich dann um 18:00 mal an, okay?
1 receives: Ja,... ich ruf Dich dann um 18:00 mal an, okay?
Input 1: okay.
2 receives: okay.
Input 2: stop
1 receives: stop

D:\Unterricht\Programmieren II\SS2007\SW 10\ChatRahmen>
```

Zur Vereinfachung werden jeweils Texte eingegeben, die auf eine einzelne Zeile passen (readLine()). Die Eingabe erfolgt abwechselnd. Sobald ein Chat-Teilnehmer ein „stop“ eingibt, wird der Chat beendet.

Teil 1
Es gibt eine zentrale Klasse MessageStore, die das Singleton Pattern implementiert. Sie stellt folgende Methoden zur Verfügung.

Method Summary

static MessageStore	**getInstance**() Implements the getInstance Methode according the Singleton Pattern.
String	**read**(short id) Returns the Message in the MessageStore.
void	**write**(String mes, short id) stores the message and the sender id in the MessageStore

Die Klasse MessageStore hat drei Attribute:
- die Instanz,
- die Nachricht
- und die Id des Aufrufers, der die letzte Message geschrieben hat. Die id in der read() und der write() Methode bezieht sich auf diese Id. So können Sie sicherstellen, dass eine Message nicht von demselben Thread abgeholt wird, der sie hineingeschrieben hat.

Hinweise zur Methode read():
- Sofern das Attribut message leer ist (keine Message vorhanden), oder die messageSenderId der Id des Aufrufers entspricht, soll wait() aufgerufen werden.
- Wenn eine Message gelesen wurde, wird das Attribut message auf null gesetzt.
- Nach Lesen einer Message wird die Methode notify() aufgerufen.

Hinweise zur Methode write():
- Wenn das Attribute message nicht null ist, muss wait() aufgerufen werden.
- Nach Schreiben einer Message wird die Methode notify() aufgerufen.

Teil 2
Es gibt eine Klasse ChatInterface, die die einzelnen Threads (implements Runnable) zur Vefügung stellt. Diese Klasse implementiert einen Konstruktor sowie die Methoden start() und run(). Sie hat drei Attribute: die Id, den Thread, sowie ein bool'sches Attribut, das angibt, ob das Objekt im lesenden oder im schreibenden Zustand ist. Der Wert der Id sowie des bool'schen Attributs wird dem Konstruktor übergeben.

Hinweise zur Methode run():

- Das Attribut reading kontrolliert, dass immer abwechselnd gelesen und geschrieben wird.
- Wenn der Chat eine Nachricht schreibt (reading == false):
  - Der User ist zu informieren, dass eine Eingabe erwartet wird.
  - Die Nachricht wird gelesen (readLine()) und geschrieben (write()).
  - Ist die Nachricht „stop", dann wird die lokale Variable stop auf true gesetzt.
  - reading wird auf true gesetzt.
- Wenn der Chat eine Nachricht liest (reading == true):
  - Die Nachricht wird gelesen und auf dem Bildschirm ausgegeben.
  - Ist die Nachricht „stop", wird die lokale Variable stop auf true gesetzt.
  - reading wird auf false gesetzt.

Teil 3
Die Klasse Chat implementiert die Methode main. Diese Klasse können Sie so übernehmen.

Teil 4
Testen Sie den Chat (von der Konsole aus).

Teil 5
Erweitern Sie die Klasse Chat, so dass sich am Chat mehr als 2 Threads gleichzeitig beteiligen. Achten Sie bitte darauf, dass nur ein Thread zu Beginn im „schreibenden" Zustand ist (false für reading). Beobachten Sie, wie der Chat abläuft. In diesem Fall beendet sich das Programm nicht sauber. Erklären Sie!

```
public class Chat
{
    public static void main (String args[])
    {
        ChatInterface first = new ChatInterface((short)1, true);
        ChatInterface second = new ChatInterface((short)2,  false);

        first.start();
        second.start();
    }
}
```

```
import java.io.*;

public class ChatInterface implements Runnable {

    private Thread thread;
    private boolean reading;
    private short id;

    public ChatInterface(short i, boolean read) {
        // initialise instance variables
        id = i;
        reading = read;
    }

    // implement the following method
    public void start() {
        if (thread == null) {
            thread = new Thread(this);
            thread.start();
        }
    }

    // implement the following method
    public void run() {
        String text;
        MessageStore message = MessageStore.getInstance();
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        PrintStream writer = new PrintStream(System.out);

        // if a message (of this or the other side) equals "stop" the boolean stop is set
to be true
        boolean stop = false;

        while (!stop) {
            try {
                if (!reading) {
                    // implement what happens, if the thread writes a message to the
other side
                    System.out.print("Input 1: ");
                    text = reader.readLine();
                    if (text.equals("stop")) {
                        stop = true;
                    } else {
                        message.write(text, id);
                    }
                } else {
                    // implement what happens, if the thread reads and prints a message
of the other side
                    text = message.read(id);
                    writer.println("2 receives: "+text);
                }
            } catch (Exception ie) {
                ie.printStackTrace();
            }

            reading = !reading;
        }
        System.exit(0);
    }
}
```

```
public final class MessageStore
{
    // instance variables - replace the example below with your own
    private String message = null;
    private static MessageStore theInstance;
    private short messageSenderId = 0;

    private MessageStore()
    {
    }
    /**
     * Returns the Message in the MessageStore.
     * @param id The id of the caller to avoid that the caller reads its own message.
     * @returns the actual message.
     */
    // implement the following method
    public synchronized String read(short id) throws InterruptedException
    {
        String returnvalue = null;

        while(id==messageSenderId || message==null)
        {
            wait();
        }

        returnvalue = message;
        message = null;

        //benachrichtig den Schreiber, dass die Message abgeholt wurde und eine neue
Nachricht geschrieben werden kann
        notify();
        return returnvalue;

    }

    /**
     * stores the message and the sender id in the MessageStore
     * @param mes the message to be stored
     * @id the sender id
     */
    // implement the following method
    public synchronized void write(String mes, short id) throws InterruptedException
    {
        while(message!=null)
        {
            wait();
        }
        this.messageSenderId = id;
        this.message = mes;

        // benachrichtig den Leser, dass eine Message gelesen werden kann
        notify();
    }

    /**
     * Implements the getInstance Methode according the Singleton Pattern.
     */
    // implement the following method
    public static synchronized MessageStore getInstance()
    {
       if(theInstance==null)
       {
           theInstance = new MessageStore();
       }
       return theInstance;
    }
}
```