tick5 submission from David Brazdil

Name	David Brazdil (db538)
College	TRINH
Submission contents	uk/ac/cam/cl/fjava/messages/RelayMessage.java uk/ac/cam/cl/fjava/messages/Execute.java uk/ac/cam/cl/fjava/messages/NewMessageType.java uk/ac/cam/cl/fjava/messages/DynamicObjectInputStream.java uk/ac/cam/cl/fjava/messages/StatusMessage.java uk/ac/cam/cl/fjava/messages/ChangeNickMessage.java uk/ac/cam/cl/fjava/messages/ChatMessage.java uk/ac/cam/cl/fjava/messages/Message.java uk/ac/cam/db538/fjava/tick5/MessageQueue.java uk/ac/cam/db538/fjava/tick5/Database.java uk/ac/cam/db538/fjava/tick5/SafeMessageQueue.java uk/ac/cam/db538/fjava/tick5/MultiQueue.java uk/ac/cam/db538/fjava/tick5/ChatServer.java uk/ac/cam/db538/fjava/tick5/ClientHandler.java
Ticker	Not yet assigned
Ticker signature	

RelayMessage.java

```
 0   package uk.ac.cam.cl.fjava.messages;
 1
 2   import java.io.Serializable;
 3   import java.util.Date;
 4
 5   public class RelayMessage extends Message implements Serializable {
 6
 7   private static final long serialVersionUID = 1L;
 8   private String from;
 9   private String message;
10
11   public RelayMessage(String from, ChatMessage original) {
12   super(original);
13   this.from = from;
14   this.message = original.getMessage();
15   }
16
17   public RelayMessage(String from, String message, Date time) {
18   super(time);
19   this.from = from;
20   this.message = message;
21   }
22
23   public String getFrom() {
24   return from;
25   }
26
27   public String getMessage() {
28   return message;
29   }
30   }
```

Execute.java

```
 0   package uk.ac.cam.cl.fjava.messages;
 1
 2   import java.lang.annotation.Retention;
 3   import java.lang.annotation.RetentionPolicy;
 4
 5   //This is an "annotation". This is explained later Workbook 2
 6   @Retention(RetentionPolicy.RUNTIME)
 7   public @interface Execute {}
```

NewMessageType.java

```
 0   package uk.ac.cam.cl.fjava.messages;
 1
 2
 3   public class NewMessageType extends Message {
 4
 5   private static final long serialVersionUID = 1L;
 6   private String name;
 7   private byte[] classData;
 8
 9   public NewMessageType(String name, byte[] classData) {
10   super();
11   this.name = name;
12   this.classData = classData;
13   }
14
15   public String getName() {
16   return name;
17   }
18
19   public byte[] getClassData() {
20   return classData;
21   }
22
23   }
```

DynamicObjectInputStream.java

```
 0  package uk.ac.cam.cl.fjava.messages;
 1
 2  import java.io.IOException;
 3  import java.io.InputStream;
 4  import java.io.ObjectInputStream;
 5  import java.io.ObjectStreamClass;
 6
 7  public class DynamicObjectInputStream extends ObjectInputStream {
 8
 9  private ClassLoader current = ClassLoader.getSystemClassLoader();
10
11  public DynamicObjectInputStream(InputStream in) throws IOException {
12  super(in);
13  }
14
15  @Override
16  protected Class<?> resolveClass(ObjectStreamClass desc) throws IOException,
17  ClassNotFoundException {
18  try {
19  return current.loadClass(desc.getName());
20  }
21  catch (ClassNotFoundException e) {
22  return super.resolveClass(desc);
23  }
24  }
25
26  public void addClass(final String name, final byte[] defn) {
27  current = new ClassLoader(current) {
28  @Override
29  protected Class<?> findClass(String className)
30  throws ClassNotFoundException {
31  if (className.equals(name)) {
32  Class<?> result = defineClass(name, defn, 0, defn.length);
33  return result;
34  } else {
35  throw new ClassNotFoundException();
36  }
37  }
38  };
39  }
40
41  }
```

StatusMessage.java

```
 0  package uk.ac.cam.cl.fjava.messages;
 1
 2  import java.io.Serializable;
 3
 4  public class StatusMessage extends Message implements Serializable {
 5
 6  private static final long serialVersionUID = 1L;
 7  private String message;
 8
 9  public StatusMessage(String message) {
10  super();
11  this.message = message;
12  }
13
14  public String getMessage() {
15  return message;
16  }
17
18  }
```

ChangeNickMessage.java

```
 0   package uk.ac.cam.cl.fjava.messages;
 1
 2   import java.io.Serializable;
 3
 4   public class ChangeNickMessage extends Message implements Serializable {
 5   private static final long serialVersionUID = 1L;
 6
 7   public String name;
 8
 9   public ChangeNickMessage(String name) {
10   super();
11   this.name = name;
12   }
13
14   }
```

ChatMessage.java

```
 0   package uk.ac.cam.cl.fjava.messages;
 1
 2   import java.io.Serializable;
 3
 4   /**
 5    * Message sent from the client to the server
 6    *
 7    */
 8   public class ChatMessage extends Message implements Serializable {
 9
10   private static final long serialVersionUID = 1L;
11   private String message;
12
13   public ChatMessage(String message) {
14   super();
15   this.message = message;
16   }
17
18   public String getMessage() {
19   return message;
20   }
21   }
```

Message.java

```
 0   package uk.ac.cam.cl.fjava.messages;
 1
 2   import java.io.Serializable;
 3   import java.util.Date;
 4
 5   public class Message implements Serializable {
 6
 7   private static final long serialVersionUID = 1L;
 8   private Date creationTime;
 9
10   public Message() {
11   creationTime = new Date();
12   }
13
14   protected Message(Message copy) {
15   creationTime = copy.creationTime;
16   }
17
18   protected Message(Date time) {
19   creationTime = time;
20   }
21
22   public Date getCreationTime() {
23   return creationTime;
24   }
25   }
```

MessageQueue.java

```
0   package uk.ac.cam.db538.fjava.tick5;
1
2   public interface MessageQueue<T> {
3   public void put(T msg);
4   public T take();
5   }
```

Database.java

```
 0  package uk.ac.cam.db538.fjava.tick5;
 1
 2  import java.sql.Connection;
 3  import java.sql.DriverManager;
 4  import java.sql.PreparedStatement;
 5  import java.sql.ResultSet;
 6  import java.sql.SQLException;
 7  import java.sql.Statement;
 8  import java.util.Date;
 9  import java.util.LinkedList;
10  import java.util.List;
11
12  import uk.ac.cam.cl.fjava.messages.RelayMessage;
13
14  public class Database {
15  private Connection connection;
16
17  public Database(String databasePath) throws SQLException {
18  try {
19  Class.forName("org.hsqldb.jdbcDriver");
20  } catch (ClassNotFoundException ex) {
21  throw new SQLException(ex);
22  }
23
24  // open connection to database
25  connection = DriverManager.getConnection("jdbc:hsqldb:file:"
26  + databasePath, "SA", "");
27
28  Statement delayStmt = connection.createStatement();
29  try {
30  delayStmt.execute("SET WRITE_DELAY FALSE");
31  } // Always update data on disk
32  finally {
33  delayStmt.close();
34  }
35
36  // turn transactions on
37  connection.setAutoCommit(false);
38
39  // create new table "messages"
40  Statement sqlStmt = connection.createStatement();
41  try {
42  sqlStmt.execute("CREATE TABLE messages(nick VARCHAR(255) NOT NULL,"
43  + "message VARCHAR(4096) NOT NULL,timeposted BIGINT NOT NULL)");
44  } catch (SQLException e) {
45  // System.out
46  // .println("Warning: Database table \"messages\" already exists.");
47  } finally {
48  sqlStmt.close();
49  }
50
51  // create new table "statistics"
52  boolean firstTimeStatistics = true;
53  sqlStmt = connection.createStatement();
54  try {
55  sqlStmt.execute("CREATE TABLE statistics(key VARCHAR(255), value INT)");
56  } catch (SQLException e) {
57  firstTimeStatistics = false;
58  // System.out
59  // .println("Warning: Database table \"statistics\" already exists.");
60  } finally {
61  sqlStmt.close();
62  }
63
64  // insert rows
65  if (firstTimeStatistics) {
66  String stmt = "INSERT INTO statistics(key, value) VALUES ('Total messages', 0)";
67  PreparedStatement insertMessage = connection.prepareStatement(stmt);
68  try {
69  insertMessage.executeUpdate();
70  } finally { // Notice use of finally clause here to finish statement
71  insertMessage.close();
```

```
 72   }
 73
 74   stmt = "INSERT INTO statistics(key, value) VALUES ('Total logins', 0)";
 75   insertMessage = connection.prepareStatement(stmt);
 76   try {
 77   insertMessage.executeUpdate();
 78   } finally { // Notice use of finally clause here to finish statement
 79   insertMessage.close();
 80   }
 81   }
 82
 83   // commit
 84   connection.commit();
 85   }
 86
 87   public void close() throws SQLException {
 88   connection.close();
 89   }
 90
 91   public void increaseLogins() throws SQLException {
 92   String stmt = "UPDATE statistics SET value = value + 1 WHERE key='Total logins'";
 93   PreparedStatement insertMessage = connection.prepareStatement(stmt);
 94   try {
 95   insertMessage.executeUpdate();
 96   } finally { // Notice use of finally clause here to finish statement
 97   insertMessage.close();
 98   }
 99
100   connection.commit();
101   }
102
103   public void addMessage(RelayMessage m) throws SQLException {
104   // insert row
105   String stmt = "INSERT INTO MESSAGES(nick,message,timeposted) VALUES (?,?,?)";
106   PreparedStatement insertMessage = connection.prepareStatement(stmt);
107   try {
108   insertMessage.setString(1, m.getFrom()); // set value of first "?"
109   insertMessage.setString(2, m.getMessage());
110   insertMessage.setLong(3, m.getCreationTime().getTime());
111   insertMessage.executeUpdate();
112   } finally { // Notice use of finally clause here to finish statement
113   insertMessage.close();
114   }
115
116   stmt = "UPDATE statistics SET value = value + 1 WHERE key='Total messages'";
117   insertMessage = connection.prepareStatement(stmt);
118   try {
119   insertMessage.executeUpdate();
120   } finally { // Notice use of finally clause here to finish statement
121   insertMessage.close();
122   }
123
124   connection.commit();
125   }
126
127   public List<RelayMessage> getRecent() throws SQLException {
128   List<RelayMessage> result = new LinkedList<RelayMessage>();
129
130   // query
131   String stmt = "SELECT nick,message,timeposted FROM messages "
132   + "ORDER BY timeposted DESC LIMIT 10";
133   PreparedStatement recentMessages = connection.prepareStatement(stmt);
134   try {
135   ResultSet rs = recentMessages.executeQuery();
136   try {
137   while (rs.next())
138   result.add(0,
139   new RelayMessage(rs.getString(1), rs.getString(2),
140   new Date(rs.getLong(3))));
141   } finally {
142   rs.close();
143   }
```

```
144   } finally {
145   recentMessages.close();
146   }
147
148   return result;
149   }
150
151   public static void main(String[] args) throws ClassNotFoundException,
152   SQLException {
153   if (args.length != 1) {
154   System.err
155   .println("Usage: java uk.ac.cam.db538.fjava.tick5.Database <database name>");
156   return;
157   }
158
159   // open connection to database
160   Class.forName("org.hsqldb.jdbcDriver");
161   Connection connection = DriverManager.getConnection("jdbc:hsqldb:file:"
162   + args[0], "SA", "");
163
164   Statement delayStmt = connection.createStatement();
165   try {
166   delayStmt.execute("SET WRITE_DELAY FALSE");
167   } // Always update data on disk
168   finally {
169   delayStmt.close();
170   }
171
172   // turn transactions on
173   connection.setAutoCommit(false);
174
175   // create new table "messages"
176   Statement sqlStmt = connection.createStatement();
177   try {
178   sqlStmt.execute("CREATE TABLE messages(nick VARCHAR(255) NOT NULL,"
179   + "message VARCHAR(4096) NOT NULL,timeposted BIGINT NOT NULL)");
180   } catch (SQLException e) {
181   System.out
182   .println("Warning: Database table \"messages\" already exists.");
183   } finally {
184   sqlStmt.close();
185   }
186
187   // insert row
188   String stmt = "INSERT INTO MESSAGES(nick,message,timeposted) VALUES (?,?,?)";
189   PreparedStatement insertMessage = connection.prepareStatement(stmt);
190   try {
191   insertMessage.setString(1, "Alastair"); // set value of first "?"
192   insertMessage.setString(2, "Hello, Andy");
193   insertMessage.setLong(3, System.currentTimeMillis());
194   insertMessage.executeUpdate();
195   } finally { // Notice use of finally clause here to finish statement
196   insertMessage.close();
197   }
198
199   // commit
200   connection.commit();
201
202   // query
203   stmt = "SELECT nick,message,timeposted FROM messages "
204   + "ORDER BY timeposted DESC LIMIT 10";
205   PreparedStatement recentMessages = connection.prepareStatement(stmt);
206   try {
207   ResultSet rs = recentMessages.executeQuery();
208   try {
209   while (rs.next())
210   System.out.println(rs.getString(1) + ": " + rs.getString(2)
211   + " [" + rs.getLong(3) + "]");
212   } finally {
213   rs.close();
214   }
215   } finally {
```

```
216   recentMessages.close();
217   }
218
219   // close connection
220   connection.close();
221   }
222   }
```

SafeMessageQueue.java

```
 0   package uk.ac.cam.db538.fjava.tick5;
 1
 2   public class SafeMessageQueue<T> implements MessageQueue<T> {
 3   private static class Link<L> {
 4   L val;
 5   Link<L> next;
 6   Link(L val) { this.val = val; this.next = null; }
 7   }
 8
 9   private Link<T> first = null;
10   private Link<T> last = null;
11
12   public synchronized void put(T val) {
13   Link<T> newLink = new Link<T>(val);
14   if (last != null)
15   last.next = newLink;
16   last = newLink;
17   if (first == null)
18   first = newLink;
19   this.notify();
20   }
21
22   public synchronized T take() {
23   while(first == null) //use a loop to block thread until data is available
24   try { this.wait(); } catch(InterruptedException ie) {}
25   Link<T> firstLink = first;
26   first = firstLink.next;
27   return firstLink.val;
28   }
29   }
```

MultiQueue.java

```
 0   package uk.ac.cam.db538.fjava.tick5;
 1
 2   import java.util.HashSet;
 3   import java.util.Set;
 4
 5   public class MultiQueue<T> {
 6   private Set<MessageQueue<T>> outputs = new HashSet<MessageQueue<T>>();
 7
 8   public void register(MessageQueue<T> q) {
 9   // add q to outputs
10   outputs.add(q);
11   }
12
13   public void deregister(MessageQueue<T> q) {
14   // remove q from outputs
15   outputs.remove(q);
16   }
17
18   public void put(T message) {
19   // copy "message" to all elements in "outputs"
20   for (MessageQueue<T> output : outputs) {
21   output.put(message);
22   }
23   }
24   }
```

ChatServer.java

```
 0  package uk.ac.cam.db538.fjava.tick5;
 1
 2  import java.io.IOException;
 3  import java.net.ServerSocket;
 4  import java.net.Socket;
 5  import java.sql.SQLException;
 6
 7  import uk.ac.cam.cl.fjava.messages.Message;
 8
 9  public class ChatServer {
10  public static void main(String args[]) {
11  // get parameter
12  int port = 0;
13  try {
14  if (args.length != 2)
15  throw new IllegalArgumentException();
16  port = Integer.parseInt(args[0]);
17  } catch (Throwable ex) {
18  System.err.println("Usage: java ChatServer <port> <database name>");
19  return;
20  }
21
22  Database database = null;
23  try {
24  database = new Database(args[1]);
25  } catch (SQLException e1) {
26  e1.printStackTrace();
27  System.err.println("Couldn't connect to the database");
28  return;
29  }
30
31  ServerSocket socket = null;
32  try {
33  socket = new ServerSocket(port);
34  } catch (IOException e) {
35  System.err.println("Cannot use port number " + port);
36  return;
37  }
38
39  MultiQueue<Message> handlers = new MultiQueue<Message>();
40
41  while (true) {
42  try {
43  Socket client = socket.accept();
44  new ClientHandler(client, handlers, database);
45  } catch (IOException e) {
46  // TODO Auto-generated catch block
47  e.printStackTrace();
48  }
49  }
50  }
51  }
```

ClientHandler.java

```
 0  package uk.ac.cam.db538.fjava.tick5;
 1
 2  import java.io.IOException;
 3  import java.io.ObjectInputStream;
 4  import java.io.ObjectOutputStream;
 5  import java.net.Socket;
 6  import java.sql.SQLException;
 7  import java.util.List;
 8  import java.util.Random;
 9
10  import uk.ac.cam.cl.fjava.messages.ChangeNickMessage;
11  import uk.ac.cam.cl.fjava.messages.ChatMessage;
12  import uk.ac.cam.cl.fjava.messages.Message;
13  import uk.ac.cam.cl.fjava.messages.RelayMessage;
14  import uk.ac.cam.cl.fjava.messages.StatusMessage;
15
16  public class ClientHandler {
17  private Socket socket;
18  private MultiQueue<Message> multiQueue;
19  private String nickname;
20  private MessageQueue<Message> clientMessages;
21  private Database database;
22
23  public ClientHandler(Socket s, MultiQueue<Message> q, Database d) {
24  socket = s;
25  multiQueue = q;
26  database = d;
27
28  clientMessages = new SafeMessageQueue<Message>();
29  multiQueue.register(clientMessages);
30
31  // get last 10 messages
32  try {
33  List<RelayMessage> recent = database.getRecent();
34  for (RelayMessage msg : recent)
35  clientMessages.put(msg);
36  } catch (SQLException ex) {
37  ex.printStackTrace();
38  System.err.println("Error while reading the database");
39  }
40
41  // increment number of logins
42  try {
43  database.increaseLogins();
44  } catch (SQLException e) {
45  e.printStackTrace();
46  System.err.println("Error while updating the database");
47  }
48
49  nickname = "Anonymous" + (new Random()).nextInt(100000);
50  multiQueue.put(new StatusMessage(nickname + " connected from "
51  + socket.getInetAddress().getHostName()));
52
53  Thread handlerInput = new Thread() {
54  @Override
55  public void run() {
56  super.run();
57
58  try {
59  ObjectInputStream stream = new ObjectInputStream(
60  socket.getInputStream());
61  while (!socket.isClosed()) {
62  Object obj = stream.readObject();
63  if (obj instanceof ChangeNickMessage) {
64  ChangeNickMessage msg = (ChangeNickMessage) obj;
65  multiQueue.put(new StatusMessage(nickname
66  + " is now known as " + msg.name));
67  nickname = msg.name;
68  } else if (obj instanceof ChatMessage) {
69  ChatMessage msg = (ChatMessage) obj;
70  RelayMessage msgR = new RelayMessage(nickname, msg);
71  multiQueue.put(msgR);
```

```
 72   try {
 73   database.addMessage(msgR);
 74   } catch (SQLException e) {
 75   e.printStackTrace();
 76   System.err
 77   .println("Error while updating the database");
 78   }
 79   }
 80   }
 81   } catch (IOException e) {
 82   multiQueue.deregister(clientMessages);
 83   multiQueue.put(new StatusMessage(nickname
 84   + " has disconnected"));
 85   } catch (ClassNotFoundException e) {
 86   e.printStackTrace();
 87   }
 88   }
 89   };
 90
 91   Thread handlerOutput = new Thread() {
 92   @Override
 93   public void run() {
 94   super.run();
 95
 96   try {
 97   ObjectOutputStream stream = new ObjectOutputStream(
 98   socket.getOutputStream());
 99   while (!socket.isClosed()) {
100   Message msg = clientMessages.take();
101   stream.writeObject(msg);
102   }
103   } catch (IOException e) {
104   e.printStackTrace();
105   }
106   }
107   };
108
109   handlerInput.setDaemon(true);
110   handlerInput.start();
111
112   handlerOutput.setDaemon(true);
113   handlerOutput.start();
114   }
115   }
```