tick4 submission from David Brazdil

Name	David Brazdil (db538)
College	TRINH
Submission contents	uk/ac/cam/cl/fjava/messages/StatusMessage.java uk/ac/cam/cl/fjava/messages/ChangeNickMessage.java uk/ac/cam/cl/fjava/messages/RelayMessage.java uk/ac/cam/cl/fjava/messages/ChatMessage.java uk/ac/cam/cl/fjava/messages/NewMessageType.java uk/ac/cam/cl/fjava/messages/Message.java uk/ac/cam/cl/fjava/messages/Execute.java uk/ac/cam/cl/fjava/messages/DynamicObjectInputStream.java uk/ac/cam/db538/fjava/tick4/SafeMessageQueue.java uk/ac/cam/db538/fjava/tick4/ChatServer.java uk/ac/cam/db538/fjava/tick4/MultiQueue.java uk/ac/cam/db538/fjava/tick4/MessageQueue.java uk/ac/cam/db538/fjava/tick4/ClientHandler.java
Ticker	Not yet assigned
Ticker signature	

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   }
```

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   }
```

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   }
```

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   }
```

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   }
```

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 {}
```

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  }
```

SafeMessageQueue.java

```
 0  package uk.ac.cam.db538.fjava.tick4;
 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  }
```

ChatServer.java

```
 0  package uk.ac.cam.db538.fjava.tick4;
 1
 2  import java.io.IOException;
 3  import java.net.ServerSocket;
 4  import java.net.Socket;
 5
 6  import uk.ac.cam.cl.fjava.messages.Message;
 7
 8  public class ChatServer {
 9  public static void main(String args[]) {
10  // get parameter
11  int port = 0;
12  try {
13  if (args.length != 1)
14  throw new IllegalArgumentException();
15  port = Integer.parseInt(args[0]);
16  } catch (Throwable ex) {
17  System.err.println("Usage: java ChatServer <port>");
18  return;
19  }
20
21  ServerSocket socket = null;
22  try {
23  socket = new ServerSocket(port);
24  } catch (IOException e) {
25  System.err.println("Cannot use port number " + port);
26  return;
27  }
28
29  MultiQueue<Message> handlers = new MultiQueue<Message>();
30
31  while (true) {
32  try {
33  Socket client = socket.accept();
34  new ClientHandler(client, handlers);
35  } catch (IOException e) {
36  // TODO Auto-generated catch block
37  e.printStackTrace();
38  }
39
40  }
41  }
42  }
```

MultiQueue.java

```
 0  package uk.ac.cam.db538.fjava.tick4;
 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  }
```

MessageQueue.java

```
0   package uk.ac.cam.db538.fjava.tick4;
1
2   public interface MessageQueue<T> {
3   public void put(T msg);
4   public T take();
5   }
```

ClientHandler.java

```
 0  package uk.ac.cam.db538.fjava.tick4;
 1
 2  import java.io.IOException;
 3  import java.io.ObjectInputStream;
 4  import java.io.ObjectOutputStream;
 5  import java.net.Socket;
 6  import java.util.Random;
 7
 8  import uk.ac.cam.cl.fjava.messages.ChangeNickMessage;
 9  import uk.ac.cam.cl.fjava.messages.ChatMessage;
10  import uk.ac.cam.cl.fjava.messages.Message;
11  import uk.ac.cam.cl.fjava.messages.RelayMessage;
12  import uk.ac.cam.cl.fjava.messages.StatusMessage;
13
14  public class ClientHandler {
15  private Socket socket;
16  private MultiQueue<Message> multiQueue;
17  private String nickname;
18  private MessageQueue<Message> clientMessages;
19
20  public ClientHandler(Socket s, MultiQueue<Message> q) {
21  socket = s;
22  multiQueue = q;
23
24  clientMessages = new SafeMessageQueue<Message>();
25  multiQueue.register(clientMessages);
26
27  nickname = "Anonymous" + (new Random()).nextInt(100000);
28  multiQueue.put(new StatusMessage(nickname + " connected from "
29  + socket.getInetAddress().getHostName()));
30
31  Thread handlerInput = new Thread() {
32  @Override
33  public void run() {
34  super.run();
35
36  try {
37  ObjectInputStream stream = new ObjectInputStream(
38  socket.getInputStream());
39  while (!socket.isClosed()) {
40  Object obj = stream.readObject();
41  if (obj instanceof ChangeNickMessage) {
42  ChangeNickMessage msg = (ChangeNickMessage) obj;
43  multiQueue.put(new StatusMessage(nickname
44  + " is now known as " + msg.name));
45  nickname = msg.name;
46  } else if (obj instanceof ChatMessage) {
47  ChatMessage msg = (ChatMessage) obj;
48  multiQueue.put(new RelayMessage(nickname, msg));
49  }
50  }
51  } catch (IOException e) {
52  multiQueue.deregister(clientMessages);
53  multiQueue.put(new StatusMessage(nickname
54  + " has disconnected"));
55  } catch (ClassNotFoundException e) {
56  e.printStackTrace();
57  }
58  }
59  };
60
61  Thread handlerOutput = new Thread() {
62  @Override
63  public void run() {
64  super.run();
65
66  try {
67  ObjectOutputStream stream = new ObjectOutputStream(
68  socket.getOutputStream());
69  while (!socket.isClosed()) {
70  Message msg = clientMessages.take();
71  stream.writeObject(msg);
```

```
72   }
73   } catch (IOException e) {
74   e.printStackTrace();
75   }
76   }
77   };
78
79   handlerInput.setDaemon(true);
80   handlerInput.start();
81
82   handlerOutput.setDaemon(true);
83   handlerOutput.start();
84   }
85   }
```