tick5 submission from Raahil Shah

Name	Raahil Shah (rds46)
College	CHURCH
Submission contents	uk/ac/cam/rds46/fjava/tick5/ChatServer.java uk/ac/cam/rds46/fjava/tick5/MultiQueue.java uk/ac/cam/rds46/fjava/tick5/MessageQueue.java uk/ac/cam/rds46/fjava/tick5/Database.java uk/ac/cam/rds46/fjava/tick5/SafeMessageQueue.java uk/ac/cam/rds46/fjava/tick5/ClientHandler.java uk/ac/cam/cl/fjava/messages/DynamicObjectInputStream.java uk/ac/cam/cl/fjava/messages/ChangeNickMessage.java uk/ac/cam/cl/fjava/messages/NewMessageType.java uk/ac/cam/cl/fjava/messages/RelayMessage.java uk/ac/cam/cl/fjava/messages/Execute.java uk/ac/cam/cl/fjava/messages/StatusMessage.java uk/ac/cam/cl/fjava/messages/ChatMessage.java uk/ac/cam/cl/fjava/messages/Message.java
Ticker	Not yet assigned
Ticker signature	

ChatServer.java

```
 0  package uk.ac.cam.rds46.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
12        //Testing
13        // args = new String[] {"1234", "db3"};
14
15         try{
16            ServerSocket ss = new ServerSocket(Integer.parseInt(args[0]));
17            MultiQueue<Message> mq = new MultiQueue<Message>();
18            Database db = new Database(args[1]);
19            while(true){
20               Socket soc = ss.accept();
21               ClientHandler ch = new ClientHandler(soc, mq, db);
22            }
23         } catch (IOException e) {
24            System.out.println("Cannot use port number " + args[0]);
25         } catch (NumberFormatException | ArrayIndexOutOfBoundsException e){
26            System.out.println("Usage: java ChatServer <port> <database-path>");
27         } catch (SQLException sqle){
28            System.out.println("Cannot create database at " + args[1]);
29         }
30      }
31  }
```

MultiQueue.java

```
 0  package uk.ac.cam.rds46.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        outputs.add(q);
10     }
11     public synchronized void deregister(MessageQueue<T> q) {
12        outputs.remove(q);
13     }
14     public synchronized void put(T message) {
15        for (MessageQueue<T> mq : outputs)
16           mq.put(message);
17     }
18  }
```

MessageQueue.java

```
 0  package uk.ac.cam.rds46.fjava.tick5;
 1
 2  public interface MessageQueue<T> {//A FIFO queue of items of type T
 3     public abstract void put(T msg); //place msg on back of queue
 4     public abstract T take(); //block until queue length >0; return head of queue
 5  }
```

Database.java

```
 0   package uk.ac.cam.rds46.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.ArrayList;
 9   import java.util.Date;
10   import java.util.List;
11
12   import uk.ac.cam.cl.fjava.messages.RelayMessage;
13
14
15   public class Database {
16
17      private Connection connection;
18
19      public Database(String path) throws SQLException {
20         // Connect to the database:
21         try {
22            Class.forName("org.hsqldb.jdbcDriver");
23         } catch (ClassNotFoundException e1) {
24            e1.printStackTrace();
25            return;
26         }
27
28         connection = DriverManager.getConnection("jdbc:hsqldb:file:"
29               + path,"SA","");
30
31         Statement delayStmt = connection.createStatement();
32         try {delayStmt.execute("SET WRITE_DELAY FALSE");}  //Always update data on disk
33         finally {delayStmt.close();}
34
35         // Turn on transaction support:
36         connection.setAutoCommit(false);
37
38         // Create messages table:
39         Statement sqlStmt = connection.createStatement();
40         try {
41            sqlStmt.execute("CREATE TABLE messages(nick VARCHAR(255) NOT NULL,"+
42                  "message VARCHAR(4096) NOT NULL,timeposted BIGINT NOT NULL)");
43         } catch (SQLException e) {
44            System.out.println("Warning: Database table \"messages\" already exists.");
45         } finally {
46            sqlStmt.close();
47         }
48
49         // Create s table:
50         Statement sqlStmt2 = connection.createStatement();
51         try {
52            sqlStmt2.execute("CREATE TABLE statistics(key VARCHAR(255),value INT)");
53            PreparedStatement insertMessage1 =
54                  connection.prepareStatement("INSERT INTO statistics(key,value) VALUES ('Total
messages',0)"),
55                  insertMessage2 =
56                  connection.prepareStatement("INSERT INTO statistics(key,value) VALUES ('Total
logins',0)");
57            try {
58               insertMessage1.executeUpdate();
59               insertMessage2.executeUpdate();
60            } finally {
61               insertMessage1.close();
62               insertMessage2.close();
63            }
64         } catch (SQLException e) {
65            System.out.println("Warning: Database table \"statistics\" already exists.");
66         } finally {
67            sqlStmt.close();
68         }
69
70         // Commit:
71         connection.commit();
```

```
72      }
73
74      public void close() throws SQLException {
75         connection.close();
76      }
77
78      public void incrementLogins() throws SQLException {
79         PreparedStatement insertMessage =
80               connection.prepareStatement("UPDATE statistics SET value = value+1 WHERE
key='Total logins'");
81         try {
82            insertMessage.executeUpdate();
83         } finally {
84            insertMessage.close();
85         }
86         connection.commit();
87      }
88
89      public void addMessage(RelayMessage m) throws SQLException {
90         // Add RelayMessage to messages table:
91         PreparedStatement insertMessage =
92               connection.prepareStatement("INSERT INTO MESSAGES(nick,message,timeposted) VALUES
(?,?,?)");
93         try {
94            insertMessage.setString(1, m.getFrom());
95            insertMessage.setString(2, m.getMessage());
96            insertMessage.setLong(3, System.currentTimeMillis());
97            insertMessage.executeUpdate();
98         } finally {
99            insertMessage.close();
100        }
101
102        // Update statistics table:
103        PreparedStatement updateMessage =
104              connection.prepareStatement("UPDATE statistics SET value = value+1 WHERE
key='Total messages'");
105        try {
106           updateMessage.executeUpdate();
107        } finally {
108           updateMessage.close();
109        }
110        connection.commit();
111
112        // Commit:
113        connection.commit();
114     }
115
116     public List<RelayMessage> getRecent() throws SQLException {
117        PreparedStatement recentMessages =
118              connection.prepareStatement("SELECT nick,message,timeposted FROM messages "
119              + "ORDER BY timeposted DESC LIMIT 10");
120        try {
121           ResultSet rs = recentMessages.executeQuery();
122           try {
123              List<RelayMessage> list = new ArrayList<RelayMessage>(10);
124              while (rs.next()) {
125                 list.add(new RelayMessage(rs.getString(1), rs.getString(2),
126                       new Date(rs.getLong(3))));
127              }
128              return list;
129           } finally {
130              rs.close();
131           }
132        } finally {
133           recentMessages.close();
134        }
135     }
136
137     public static void main(String[] args) {
138        // Testing
139        // args = new String[] {"/Users/Raahil/Desktop/chat-database"};
140
141        try {
142           // Connect to the database:
143           Class.forName("org.hsqldb.jdbcDriver");
```

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

SafeMessageQueue.java

```
 0  package uk.ac.cam.rds46.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     private Link<T> first = null;
 9     private Link<T> last = null;
10
11     public synchronized void put(T val) {
12        Link<T> bot = new Link<T>(val);
13        if (first == null) first = bot;
14        else last.next = bot;
15        last = bot;
16        this.notify();
17     }
18
19     public synchronized T take() {
20        while(first == null) // use a loop to block thread until data is available
21           try {this.wait();} catch(InterruptedException ie) {}
22        T val = first.val;
23        first = first.next;
24        if (first == null) last = first;
25        return val;
26     }
27  }
```

ClientHandler.java

```
 0  package uk.ac.cam.rds46.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 Database database;
20     private String nickname;
21     private MessageQueue<Message> clientMessages;
22
23     public ClientHandler(Socket s, MultiQueue<Message> q, Database db) {
24        socket = s;
25        multiQueue = q;
26        database = db;
27        clientMessages = new SafeMessageQueue<Message>();
28        multiQueue.register(clientMessages);
29
30        try {
31           database.incrementLogins();
32           List<RelayMessage> list = database.getRecent();
33           for (int i = list.size() - 1; i >= 0; i--) {
34              clientMessages.put(list.get(i));
35           }
36        } catch (SQLException e) {
37           e.printStackTrace();
38        }
39
40        nickname = "Anonymous" + (new Random()).nextInt(100000);
41        StatusMessage connectionMsg = new StatusMessage(nickname + " connected from "
42           + socket.getInetAddress().getCanonicalHostName() + ".");
43        multiQueue.put(connectionMsg);
44
45        Thread incomingHandlerThrd = new Thread() {
46           @Override
47           public void run() {
48              try {
49                 ObjectInputStream ois = new ObjectInputStream(socket.getInputStream());
50                 while (true) {
51                    try {
52                       Object msg = ois.readObject();
53                       if (msg instanceof ChangeNickMessage) {
54                          String newNick = ((ChangeNickMessage) msg).name;
55                          nickname = newNick;
56                          StatusMessage newNickStatus = new StatusMessage(nickname + " is now
known as " + newNick + ".");
57                          multiQueue.put(newNickStatus);
58                       } else if (msg instanceof ChatMessage) {
59                          RelayMessage relay = new RelayMessage(nickname, (ChatMessage) msg);
60                          multiQueue.put(relay);
61                          try {
62                             database.addMessage(relay);
63                          } catch (SQLException e) {
64                             e.printStackTrace();
65                          }
66                                                 }
67                    } catch (ClassNotFoundException e) {
68                       e.printStackTrace();
69                    }
70                 }
71              } catch (IOException e) {
```

```
72                  StatusMessage dcStatus = new StatusMessage(nickname + " has disconnected.");
73                  multiQueue.put(dcStatus);
74                  multiQueue.deregister(clientMessages);
75                  return;
76               }
77            }
78
79         };
80         incomingHandlerThrd.setDaemon(true);
81         incomingHandlerThrd.start();
82
83         Thread ourgoingHandlerThrd = new Thread() {
84            @Override
85            public void run() {
86               try {
87                  ObjectOutputStream oos = new ObjectOutputStream(socket.getOutputStream());
88                  while (true)
89                     oos.writeObject(clientMessages.take());
90               } catch (IOException e) {
91                  e.printStackTrace();
92                  return;
93               }
94            }
95         };
96         ourgoingHandlerThrd.setDaemon(true);
97         ourgoingHandlerThrd.start();
98      }
99
100  }
```

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  }
```

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  }
```

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  }
```

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 {}
```

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  }
```

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  }
```

tick4 submission from Raahil Shah

Name	Raahil Shah (rds46)
College	CHURCH
Submission contents	uk/ac/cam/cl/fjava/messages/DynamicObjectInputStream.java uk/ac/cam/cl/fjava/messages/ChangeNickMessage.java uk/ac/cam/cl/fjava/messages/NewMessageType.java uk/ac/cam/cl/fjava/messages/RelayMessage.java uk/ac/cam/cl/fjava/messages/Execute.java uk/ac/cam/cl/fjava/messages/StatusMessage.java uk/ac/cam/cl/fjava/messages/ChatMessage.java uk/ac/cam/cl/fjava/messages/Message.java uk/ac/cam/rds46/fjava/tick4/MultiQueue.java uk/ac/cam/rds46/fjava/tick4/SafeMessageQueue.java uk/ac/cam/rds46/fjava/tick4/MessageQueue.java uk/ac/cam/rds46/fjava/tick4/ClientHandler.java uk/ac/cam/rds46/fjava/tick4/ChatServer.java
Ticker	Not yet assigned
Ticker signature	

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  }
```

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  }
```

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  }
```

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 {}
```

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  }
```

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  }
```

MultiQueue.java

```
 0  package uk.ac.cam.rds46.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        outputs.add(q);
10     }
11     public synchronized void deregister(MessageQueue<T> q) {
12        outputs.remove(q);
13     }
14     public synchronized void put(T message) {
15        for (MessageQueue<T> mq : outputs)
16           mq.put(message);
17     }
18  }
```

SafeMessageQueue.java

```
0   package uk.ac.cam.rds46.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      private Link<T> first = null;
9      private Link<T> last = null;
10
11     public synchronized void put(T val) {
12        Link<T> bot = new Link<T>(val);
13        if (first == null) first = bot;
14        else last.next = bot;
15        last = bot;
16        this.notify();
17     }
18
19     public synchronized T take() {
20        while(first == null) // use a loop to block thread until data is available
21           try {this.wait();} catch(InterruptedException ie) {}
22        T val = first.val;
23        first = first.next;
24        if (first == null) last = first;
25        return val;
26     }
27  }
```

MessageQueue.java

```
0   package uk.ac.cam.rds46.fjava.tick4;
1
2   public interface MessageQueue<T> {//A FIFO queue of items of type T
3      public abstract void put(T msg); //place msg on back of queue
4      public abstract T take(); //block until queue length >0; return head of queue
5   }
```

ClientHandler.java

```
 0  package uk.ac.cam.rds46.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        StatusMessage connectionMsg = new StatusMessage(nickname + " connected from "
29           + socket.getInetAddress().getCanonicalHostName() + ".");
30        multiQueue.put(connectionMsg);
31
32        Thread incomingHandlerThrd = new Thread() {
33           @Override
34           public void run() {
35              try {
36                 ObjectInputStream ois = new ObjectInputStream(socket.getInputStream());
37                 while (true) {
38                    try {
39                       Object msg = ois.readObject();
40                       if (msg instanceof ChangeNickMessage) {
41                          String newNick = ((ChangeNickMessage) msg).name;
42                          nickname = newNick;
43                          StatusMessage newNickStatus = new StatusMessage(nickname + " is now
known as " + newNick + ".");
44                          multiQueue.put(newNickStatus);
45                       } else if (msg instanceof ChatMessage) {
46                          RelayMessage relay = new RelayMessage(nickname, (ChatMessage) msg);
47                          multiQueue.put(relay);
48                       }
49                    } catch (ClassNotFoundException e) {
50                       e.printStackTrace();
51                    }
52                 }
53              } catch (IOException e) {
54                 StatusMessage dcStatus = new StatusMessage(nickname + " has disconnected.");
55                 multiQueue.put(dcStatus);
56                 multiQueue.deregister(clientMessages);
57                 return;
58              }
59           }
60
61        };
62        incomingHandlerThrd.setDaemon(true);
63        incomingHandlerThrd.start();
64
65        Thread ourgoingHandlerThrd = new Thread() {
66           @Override
67           public void run() {
68              try {
69                 ObjectOutputStream oos = new ObjectOutputStream(socket.getOutputStream());
70                 while (true)
71                    oos.writeObject(clientMessages.take());
```

```
72              } catch (IOException e) {
73                 e.printStackTrace();
74                 return;
75              }
76           }
77        };
78        ourgoingHandlerThrd.setDaemon(true);
79        ourgoingHandlerThrd.start();
80     }
81
82  }
```

ChatServer.java

```
 0  package uk.ac.cam.rds46.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
11        // Testing
12        // args = new String[] {"1234"};
13
14         try{
15            ServerSocket ss = new ServerSocket(Integer.parseInt(args[0]));
16            MultiQueue<Message> mq = new MultiQueue<Message>();
17            while(true){
18               Socket soc = ss.accept();
19               ClientHandler ch = new ClientHandler(soc, mq);
20            }
21         }
22         catch (IOException e) {
23            System.out.println("Cannot use port number " + args[0]);
24         }
25         catch (NumberFormatException | ArrayIndexOutOfBoundsException e){
26            System.out.println("Usage: java ChatServer <port>");
27         }
28      }
29  }
```

tick3 submission from Raahil Shah

Name	Raahil Shah (rds46)
College	CHURCH
Submission contents	uk/ac/cam/rds46/fjava/tick3/SafeMessageQueue.java uk/ac/cam/rds46/fjava/tick3/UnsafeMessageQueue.java uk/ac/cam/rds46/fjava/tick3/QueueTest.java uk/ac/cam/rds46/fjava/tick3/ProducerConsumer.java uk/ac/cam/rds46/fjava/tick3/MessageQueue.java uk/ac/cam/rds46/fjava/tick3/BankSimulator.java
Ticker	Not yet assigned
Ticker signature	

SafeMessageQueue.java

```
 0  package uk.ac.cam.rds46.fjava.tick3;
 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     private Link<T> first = null;
 9     private Link<T> last = null;
10
11     public synchronized void put(T val) {
12        Link<T> bot = new Link<T>(val);
13        if (first == null) first = bot;
14        else last.next = bot;
15        last = bot;
16        this.notify();
17     }
18
19     public synchronized T take() {
20        while(first == null) // use a loop to block thread until data is available
21           try {this.wait();} catch(InterruptedException ie) {}
22        T val = first.val;
23        first = first.next;
24        if (first == null) last = first;
25        return val;
26     }
27  }
```

UnsafeMessageQueue.java

```
 0  package uk.ac.cam.rds46.fjava.tick3;
 1
 2  public class UnsafeMessageQueue<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     private Link<T> first = null;
 9     private Link<T> last = null;
10
11     public void put(T val) {
12        Link<T> bot = new Link<T>(val);
13        if (first == null) first = bot;
14        else last.next = bot;
15        last = bot;
16     }
17
18     public T take() {
19        while(first == null) //use a loop to block thread until data is available
20           try {Thread.sleep(100);} catch(InterruptedException ie) {}
21        T val = first.val;
22        first = first.next;
23        if (first == null) last = first;
24        return val;
25     }
26  }
```

QueueTest.java

```
 0  package uk.ac.cam.rds46.fjava.tick3;
 1
 2  public class QueueTest {
 3
 4     private class Producer extends Thread {
 5        private int sent = 0;
 6        public void run() {
 7           for (int i = 0; i < 50000; ++i) {
 8              q.put("" + i);
 9              sent++;
10           }
11        }
12        public int numberProduced() {return sent;}
13     }
14
15     private class Consumer extends Thread {
16        private int recv = 0;
17        public void run() {
18           while (!q.take().equals("EOF")) {
19              recv++;
20           }
21           q.put("EOF");
22        }
23        public int numberConsumed() {return recv;}
24     }
25
26     private MessageQueue<String> q;
27     private Consumer[] consumers;
28     private Producer[] producers;
29
30     QueueTest(MessageQueue<String> q, int c, int p) {
31        this.q = q;
32        consumers = new Consumer[c];
33        for (int i = 0; i < c; ++i)
34           consumers[i] = new Consumer();
35        producers = new Producer[p];
36        for (int i = 0; i < p; ++i)
37           producers[i] = new Producer();
38     }
39
40     public void run() {
41
42        for (Consumer c : consumers) c.start();
43        for (Producer p : producers) p.start();
44        for (Producer p : producers) try {p.join();} catch (InterruptedException e) {}
45
46        q.put("EOF");
47        //terminate join at 10 secs since EOF marker may get lost
48        for (Consumer c : consumers) try {c.join(10000);} catch (InterruptedException e) {}
49
50        int recv = 0;
51        for (Consumer consumer : consumers) recv += consumer.numberConsumed();
52        int sent = 0;
53        for (Producer p : producers) sent += p.numberProduced();
54        System.out.println(recv + " / " + sent);
55     }
56
57     public static void main(String[] args) {
58  //      System.out.println("** UNSAFE ** ");
59  //      new QueueTest(new UnsafeMessageQueue<String>(), 1, 1).run();
60  //      new QueueTest(new UnsafeMessageQueue<String>(), 3, 1).run();
61  //      new QueueTest(new UnsafeMessageQueue<String>(), 1, 3).run();
62  //      new QueueTest(new UnsafeMessageQueue<String>(), 3, 3).run();
63
64        System.out.println("** SAFE ** ");
65        new QueueTest(new SafeMessageQueue<String>(), 1, 1).run();
66        new QueueTest(new SafeMessageQueue<String>(), 3, 1).run();
67        new QueueTest(new SafeMessageQueue<String>(), 1, 3).run();
68        new QueueTest(new SafeMessageQueue<String>(), 3, 3).run();
69     }
70  }
```

ProducerConsumer.java

```
 0  package uk.ac.cam.rds46.fjava.tick3;
 1
 2  class ProducerConsumer {
 3     private MessageQueue<Character> m = new UnsafeMessageQueue<Character>();
 4     private class Producer implements Runnable {
 5        char[] cl = "Computer Laboratory".toCharArray();
 6        public void run() {
 7           for (int i = 0; i < cl.length; i++) {
 8              m.put(cl[i]);
 9              try {Thread.sleep(500);} catch (InterruptedException e) {
10                 e.printStackTrace();
11              }
12           }
13        }
14     }
15     private class Consumer implements Runnable {
16        public void run() {
17           while (true) {
18              System.out.print(m.take());
19              System.out.flush();
20           }
21        }
22     }
23     void execute() {
24        new Thread(new Producer()).start();
25        new Thread(new Consumer()).start();
26     }
27     public static void main(String[] args) {
28        new ProducerConsumer().execute();
29     }
30  }
```

MessageQueue.java

```
 0  package uk.ac.cam.rds46.fjava.tick3;
 1
 2  public interface MessageQueue<T> {//A FIFO queue of items of type T
 3     public abstract void put(T msg); //place msg on back of queue
 4     public abstract T take(); //block until queue length >0; return head of queue
 5  }
```

BankSimulator.java

```
 0  package uk.ac.cam.rds46.fjava.tick3;
 1
 2  import java.util.Random;
 3
 4  public class BankSimulator {
 5
 6     private class BankAccount {
 7        private int balance;
 8        private int acc;
 9        BankAccount(int accountNumber, int deposit) {
10           balance = deposit;
11           acc = accountNumber;
12        }
13        public int getAccountNumber() {
14           return acc;
15        }
16
17        public void transferTo(BankAccount b, int amount) {
18           BankAccount less = b, more = this;
19           if (b.acc < this.acc) { more = b; less = this; }
20           synchronized (less) {
21              synchronized (more) {
22                 balance -= amount;
23                 b.balance += amount;
24              }
25           }
26        }
27     }
28
29     private static Random r = new Random();
30     private class RoboTeller extends Thread {
31        public void run() {
32           //Robots work from 9am until 5pm; one customer per second
33           for(int i=9*60*60; i<17*60*60; i++) {
34              int a = r.nextInt(account.length);
35              int b = r.nextInt(account.length);
36              account[a].transferTo(account[b], r.nextInt(100));
37           }
38        }
39     }
40
41     private int capital;
42     private BankAccount[] account;
43     private RoboTeller[] teller;
44
45     public BankSimulator(int capital, int accounts, int tellers) {
46         this.capital = capital;
47         this.account = new BankAccount[accounts];
48         this.teller = new RoboTeller[tellers];
49        for(int i=0; i<account.length; i++)
50           account[i]  = new BankAccount(i,capital/account.length);
51      }
52
53     public int getCapital() {return capital;}
54
55     public void runDay() {
56        for(int i=0; i<teller.length; i++)
57           teller[i] = new RoboTeller();
58        for(int i=0; i<teller.length; i++)
59           teller[i].start();
60
61        int done = 0;
62        while(done < teller.length)
63           try{teller[done].join();done++;} catch(InterruptedException e) {}
64
65        int finalCapital = 0;
66        for(int i=0; i<account.length; i++)
67        finalCapital += account[i].balance;
68        capital = finalCapital;
69     }
70
71     public static void main(String[] args) {
```

```
72          BankSimulator javaBank = new BankSimulator(10000,10,100);
73          javaBank.runDay();
74          System.out.println("Capital at close: £"+javaBank.getCapital());
75      }
76  }
```

	Thread S			Shared		Thread T		
	`a.transferTo(b,10)`	`tmp1`	`tmp2`	`a.bal`	`b.bal`	`b.transferTo(a,20)`	`tmp1`	`tmp2`
1				100	100			
2	`tmp1 = a.bal-10`	90		100	100			
3		90	100	100	100	`tmp1 = b.bal-20`	80	
4		90	100	100	80	`b.bal = tmp1`	80	
5	`a.bal = tmp1`	90	100	90	80		80	
6	`tmp2 = b.bal+10`	90	90	90	80		80	
7	`b.bal = tmp2`	90	90	90	90		80	
8		90	90	90	90	`tmp2 = a.bal+20`	80	110
9		90	90	110	90	`a.bal = tmp2`	80	110
10		90	90	110	90		80	110