

# Hoops Code Inspection

Reviewer Name	
Preparation Time (minutes)	
# Major Issues (user-visible bugs)	
# Minor Issues	
# Other Issues	

**Total lines of code: 450**

## Author Comments

This inspection covers the core classes for the "Hoops" servlet application.

Thanks for your help.

# Inspection Files

<b><u>ConfigurationException.java</u></b> .....	1
<b><u>GameStatus.java</u></b> .....	2
<b><u>HoopsServlet.java</u></b> .....	5
<b><u>MailUtils.java</u></b> .....	7
<b><u>NotificationManager.java</u></b> .....	8
<b><u>ScheduleManager.java</u></b> .....	9

## ConfigurationException.java

```
1  /*      $Workfile: $
2  $Modtime: $
3  $Author: $
4
5  Originally created on July 25, 2002 by Eric Smith.
6
7  Copyright (c) 2002, Eric Smith. All rights reserved.
8  */
9
10 package ericsmith.hoops;
11
12
13
14 public class ConfigurationException extends Exception
15 {
16     public ConfigurationException(String message)
17     {
18         super(message);
19     }
20 }
```

## GameStatus.java

```
1  /*
2  * $Workfile: $
3  * $Modtime: $
4  * $Author: $
5  * Originally created on July 24, 2002 by Eric Smith.
6  * Copyright (c) 2002, Eric Smith. All rights reserved.
7  */
8  package ericsmith.hoops;
9
10 import ericsmith.util.FileUtils;
11 import java.io.File;
12 import java.io.FileInputStream;
13 import java.io.FileNotFoundException;
14 import java.io.FileOutputStream;
15 import java.io.IOException;
16 import java.io.OutputStream;
17 import java.util.Calendar;
18 import java.util.Iterator;
19 import java.util.List;
20 import org.jdom.Document;
21 import org.jdom.Element;
22 import org.jdom.JDOMException;
23 import org.jdom.input.SAXBuilder;
24 import org.jdom.output.XMLOutputter;
25 import org.jdom.transform.JDOMSource;
26 import javax.xml.transform.Transformer;
27 import javax.xml.transform.TransformerException;
28 import javax.xml.transform.TransformerFactory;
29 import javax.xml.transform.stream.StreamSource;
30 import javax.xml.transform.stream.StreamResult;
31
32
33 /**
34  * Represents the status for a game.
35  *
36  * @author Eric
37  * @created July 25, 2002
38  */
39 public class GameStatus
40 {
41     private final static String GAME_TEMPLATE = "game-template.xml";
42
43     private Document m_doc;
44     private File m_gameFile;
45
46
47     /**
48      * Constructor for the GameStatus object. Clients must use the getGame
49      * factory method to create instances.
50      */
51     private GameStatus() { }
52
53
54     /**
55      * Constructor for the GameStatus object. Clients must use the getGame
56      * factory method to create instances.
57      *
58      * @param data XML file containing game data.
59      * @param isNew indicates if this is a new game, or existing.
60      * @exception ConfigurationException If there is a problem with the game
61      *         template file.
62      */
63     private GameStatus(File data, boolean isNew, String date) throws ConfigurationException
64     {
65         m_gameFile = data;
66
67         SAXBuilder builder = new SAXBuilder();
68
69         try
70         {
71             m_doc = builder.build(new FileInputStream(data));
72
73             if (isNew)
74             {
75                 addDate(date);
76             }
77         }
78         catch (FileNotFoundException ex)
79         {
80             throw new ConfigurationException(ex.getMessage());
81         }
82         catch (JDOMException ex)
83         {
84             throw new ConfigurationException(ex.getMessage());
85         }
86     }
87
88
89     /**
90      * Adds the given date to the current game document.
91      */
92     private void addDate(String dateStr)
93     {
94         Element game = m_doc.getRootElement();
```

```

95     Element details = game.getChild("details");
96     Element date = new Element("date");
97     date.setText(dateStr);
98     details.addContent(date);
99 }
100
101
102 /**
103  * Factory method that gets the game status for the given day's game. If
104  * the game status doesn't exist, it is created.
105  *
106  * @param date the day to retrieve game data for.
107  * @param configDir directory to retrieve configuration info from.
108  * @param dataDir directory to store and retrieve data files from.
109  * @return The game
110  * @exception ConfigurationException If there is a problem with the game
111  *         template file.
112  */
113 public static GameStatus getGame(String date, String configDir, String dataDir) throws ConfigurationExcept «
ion
114 {
115     String gameFileName = date + ".xml";
116     File gameFile = new File(dataDir + gameFileName);
117
118     try
119     {
120         if (gameFile.exists())
121         {
122             return new GameStatus(gameFile, false, date);
123         }
124         else
125         {
126             FileUtils.copy(configDir + GAME_TEMPLATE, dataDir + gameFileName);
127             return new GameStatus(gameFile, true, date);
128         }
129     }
130     catch (IOException ex)
131     {
132         throw new ConfigurationException(ex.getMessage());
133     }
134 }
135
136
137 /**
138  * Writes the game status as XML.
139  *
140  * @param out destination stream.
141  * @exception IOException If there is a problem writing to the supplied stream.
142  */
143 public void serialize(OutputStream out) throws IOException
144 {
145     XMLOutputter outputter = new XMLOutputter();
146     outputter.output(m_doc, out);
147 }
148
149
150 /**
151  * Writes the game status as the result of a transformation.
152  *
153  * @param name name of the currently logged-in player.
154  * @param style path to the style sheet to transform with.
155  * @param out destination stream.
156  * @exception IOException If there is a problem writing to the supplied stream.
157  */
158 public void transform(String name, String style, OutputStream out) throws IOException, TransformerExceptio «
n
159 {
160     Transformer transformer = TransformerFactory.newInstance().newTransformer(new StreamSource(style));
161     transformer.setParameter("current-player", name);
162     transformer.transform(new JDOMSource(m_doc), new StreamResult(out));
163 }
164
165
166 /**
167  * Saves the game data to a file to keep it persistent.
168  *
169  * @throws IOException If there is a problem writing the file.
170  */
171 public void save() throws IOException
172 {
173     XMLOutputter outputter = new XMLOutputter();
174     FileOutputStream out = null;
175     try
176     {
177         out = new FileOutputStream(m_gameFile);
178         outputter.output(m_doc, out);
179     }
180     finally
181     {
182         if (out != null)
183         {
184             out.close();
185         }
186     }
187 }
188
189

```

```

190  /**
191  * Sets the game status for a particular player.
192  *
193  * @param name the player's name
194  * @param status the player's status.
195  * @param comment comment from the player.
196  */
197  public void setPlayerStatus(String name, String status, String comment)
198  {
199      if (status == null)
200      {
201          status = "";
202      }
203
204      if (comment == null)
205      {
206          comment = "";
207      }
208
209      Element game = m_doc.getRootElement();
210      Element players = game.getChild("players");
211
212      Element player = getPlayerElement(players, name);
213
214      Element playerStatus = player.getChild("status");
215      playerStatus.setText(status);
216
217      Element playerComment = player.getChild("comment");
218      playerComment.setText(comment);
219  }
220
221  /**
222  * Gets a player element by name. If the element doesn't exist, it is created
223  * and added to the players element.
224  *
225  * @param players the players element containing the player.
226  * @param name the name of the player to retrieve.
227  * @return the player element.
228  */
229  private Element getPlayerElement(Element players, String name)
230  {
231      List playerList = players.getChildren();
232      Iterator iter = playerList.iterator();
233      while (iter.hasNext())
234      {
235          Element player = (Element) iter.next();
236          if (player.getChildTextNormalize("name").equals(name))
237          {
238              return player;
239          }
240      }
241
242      Element player = new Element("player");
243
244      Element playerName = new Element("name");
245      playerName.setText(name);
246      player.addContent(playerName);
247
248      Element playerStatus = new Element("status");
249      player.addContent(playerStatus);
250
251      Element playerComment = new Element("comment");
252      player.addContent(playerComment);
253
254      players.addContent(player);
255
256      return player;
257  }
258 }
259
260

```

## HoopsServlet.java

```
1  /*      $Workfile: $
2         $Modtime: $
3         $Author: $
4
5         Originally created on July 25, 2002 by Eric Smith.
6
7         Copyright (c) 2002, Eric Smith. All rights reserved.
8  */
9
10 package ericsmith.hoops;
11
12 import java.io.IOException;
13 import java.io.PrintWriter;
14 import java.util.Enumeration;
15 import javax.servlet.ServletException;
16 import javax.servlet.http.HttpServlet;
17 import javax.servlet.http.HttpServletRequest;
18 import javax.servlet.http.HttpServletResponse;
19 import javax.servlet.http.HttpSession;
20 import javax.servlet.ServletConfig;
21 import javax.xml.transform.TransformerException;
22
23
24 /**
25  * @author Eric
26  * @created July 13, 2002
27  */
28 public final class HoopsServlet extends HttpServlet
29 {
30     private final static String DATA_DIR = "WEB-INF\\data\\";
31     private final static String CONFIG_DIR = "WEB-INF\\config\\";
32     private final static String STYLE_SHEET = "status.xsl";
33
34     private SchedulerManager schedulerManager;
35     private NotificationManager m_notificationManager;
36
37
38     /**
39      * Initialize the servlet before any requests come through.
40      */
41     public void init(ServletConfig config) throws ServletException
42     {
43         super.init(config);
44         schedulerManager = new SchedulerManager(config);
45         m_notificationManager = new NotificationManager(
46             getServletContext().getRealPath("/") + CONFIG_DIR,
47             getServletContext().getRealPath("/") + DATA_DIR);
48     }
49
50
51     /**
52      * Respond to a GET request for the content produced by this servlet.
53      *
54      * @param request The servlet request we are processing
55      * @param response The servlet response we are producing
56      * @exception IOException if an input/output error occurs
57      * @exception ServletException if a servlet error occurs
58      */
59     public void doGet(HttpServletRequest request, HttpServletResponse response)
60         throws IOException, ServletException
61     {
62         response.setContentType("text/html");
63
64         try
65         {
66             if (!schedulerManager.isGameScheduledToday())
67             {
68                 // Send 'no game today' page.
69                 response.sendRedirect("no-game.html");
70                 return;
71             }
72
73             String playerName = getPlayer(request);
74             if (playerName == null)
75             {
76                 // Send login page.
77                 response.sendRedirect("login.html");
78                 return;
79             }
80
81             GameStatus game = GameStatus.getGame(schedulerManager.getTodayString(),
82                 getServletContext().getRealPath("/") + CONFIG_DIR,
83                 getServletContext().getRealPath("/") + DATA_DIR);
84
85             String status = request.getParameter("ps");
86             String comment = request.getParameter("pc");
87             if (status != null || comment != null)
88             {
89                 game.setPlayerStatus(playerName, status, comment);
90             }
91
92             if (request.getParameter("raw") != null)
93             {
94
```

```

95         // Return the XML game data to the response stream.
96         response.setContentType("text/xml");
97         game.serialize(response.getOutputStream());
98     }
99     else
100     {
101         game.transform(playerName, getServletContext().getRealPath("/") + STYLE_SHEET, response.getOut «
putStream());
102     }
103
104     game.save();
105 }
106 catch (ConfigurationException ex)
107 {
108     throw new ServletException(ex.getMessage());
109 }
110 catch (TransformerException ex)
111 {
112     throw new ServletException("There was a problem transforming the output with " + STYLE_SHEET + ".\ «
n" + ex.getMessage());
113 }
114 }
115 }
116 }
117
118 /**
119  * Gets the name of the logged-in player, if known.
120  * @return The player's name, or null if it is not known.
121  */
122 private String getPlayer(HttpServletRequest request)
123 {
124     // Honor logout request first.
125     if (request.getParameter("logout") != null && !request.getSession().isNew())
126     {
127         request.getSession().invalidate();
128     }
129
130     HttpSession session = request.getSession();
131     String playerName = (String)session.getAttribute("name");
132     if (playerName == null || playerName.equals(""))
133     {
134         playerName = request.getParameter("pn");
135         if (playerName != null && !playerName.equals(""))
136         {
137             session.setAttribute("name", playerName);
138         }
139     }
140
141     if (playerName != null && playerName.equals(""))
142     {
143         playerName = null;
144     }
145
146     return playerName;
147 }
148 }
149 }
150

```



## MailUtils.java

```
1  /*      $Workfile: $
2  $Modtime: $
3  $Author: $
4
5  Originally created on September 21, 2002 by Eric Smith.
6
7  Copyright (c) 2002, Eric Smith. All rights reserved.
8  */
9
10 package ericsmith.hoops;
11
12
13 public class MailUtils
14 {
15     /** SMTP mail server */
16     private String m_mailServer;
17
18
19     /**
20      * Sends an HTML e-mail message.
21      *
22      * @param content The HTML content of the mail message.
23      */
24     public void sendMessageHTML(String content)
25     {
26         Properties mailProps = System.getProperties();
27         mailProps.put("mail.smtp.host", m_mailServer);
28         Session session = Session.getInstance(mailProps, null);
29         Transport tr = session.getTransport("smtp");
30         tr.connect(m_mailServer, smtpUserName, smtpPassword);
31         MimeMessage message = new MimeMessage(session);
32         message.setFrom(new InternetAddress(from));
33         message.setRecipients(Message.RecipientType.TO, InternetAddress.parse(to));
34         message.setSubject(subject);
35
36         message.setContent(content, "text/html");
37
38         if (useAccount)
39         {
40             tr.send(message);
41             tr.close();
42         }
43         else
44         {
45             Transport.send(message);
46         }
47     }
48
49
50     /**
51      * Sends a plain text e-mail message.
52      *
53      * @param content The text content of the mail message.
54      */
55     public void sendMessageText(String content)
56     {
57     }
58
59
60
61 }
```

## NotificationManager.java

```
1  /*      $Workfile: $
2  $Modtime: $
3  $Author: $
4
5  Originally created on September 20, 2002 by Eric Smith.
6
7  Copyright (c) 2002, Eric Smith. All rights reserved.
8  */
9
10 package ericsmith.hoops;
11
12 import java.util.Date;
13 import java.util.Timer;
14 import java.util.TimerTask;
15
16
17 /**
18  * Class to manage e-mail notifications to players.
19  */
20 public class NotificationManager
21 {
22     /** Number of milliseconds in a day. */
23     private static final int ONE_DAY = 10000; //86400000
24
25     /** Directory where configuration data is stored. */
26     private String m_configDir;
27
28     /** Directory where application data is stored. */
29     private String m_dataDir;
30
31     /** A timer for sending out e-mail invitations to a day's game. */
32     private Timer m_inviteTimer = new Timer(true);
33
34
35     /**
36     * Constructor.
37     */
38     public NotificationManager(String configDir, String dataDir)
39     {
40         m_configDir = configDir;
41         m_dataDir = dataDir;
42
43         m_inviteTimer.scheduleAtFixedRate(new InviteTask(), new Date(), ONE_DAY);
44     }
45
46
47     /**
48     * Sends an e-mail invitation to log in to the system.
49     */
50     public void sendInvitation()
51     {
52     }
53 }
54
55
56
57 /**
58  * TimerTask that sends out game invitations.
59  */
60 class InviteTask extends TimerTask
61 {
62     public void run()
63     {
64         System.out.println("Do yo want to play basketball?");
65     }
66 }
```

## ScheduleManager.java

```
1  /*      $Workfile: $
2  $Modtime: $
3  $Author: $
4
5  Originally created on September 5, 2002 by Eric Smith.
6
7  Copyright (c) 2002, Eric Smith. All rights reserved.
8  */
9
10 package ericsmith.hoops;
11
12
13 import java.util.ArrayList;
14 import java.util.Calendar;
15 import java.util.Date;
16 import java.util.List;
17 import java.text.SimpleDateFormat;
18 import java.text.ParseException;
19 import java.util.StringTokenizer;
20 import javax.servlet.ServletConfig;
21
22
23
24 /**
25  * Abstracts the game scheduling.
26  */
27 public class ScheduleManager
28 {
29     /** The length of an ISO 8601 date */
30     private static final int ISO8601_LEN = 10;
31
32     private String m_schedule;
33     private ArrayList m_additions = new ArrayList();
34     private ArrayList m_cancellations = new ArrayList();
35
36
37     /**
38     * Constructor.
39     *
40     * @param config The Servlet configuration from which to read schedule data.
41     */
42     public ScheduleManager(ServletConfig config)
43     {
44         m_schedule = config.getInitParameter("schedule").toLowerCase();
45         String additions = config.getInitParameter("schedule-additions");
46         if (additions != null)
47         {
48             parseExceptions(additions, m_additions);
49         }
50
51         String cancellations = config.getInitParameter("schedule-cancellations");
52         if (cancellations != null)
53         {
54             parseExceptions(cancellations, m_cancellations);
55         }
56     }
57
58
59     /**
60     * Parses an exception list and adds the exceptions to a list.
61     *
62     * @param s The exception list to parse.
63     * @param list The list to add exceptions to.
64     */
65     private void parseExceptions(String s, List list)
66     {
67         SimpleDateFormat dateFormat = new SimpleDateFormat();
68
69         StringTokenizer st = new StringTokenizer(s, "|");
70         while (st.hasMoreElements())
71         {
72             String ex = st.nextToken();
73
74             try
75             {
76                 ScheduleException exception =
77                     new ScheduleException(dateFormat.parse(ex.substring(0, ISO8601_LEN)),
78                     ex.substring(ISO8601_LEN));
79
80                 list.add(exception);
81             }
82             catch (ParseException e)
83             {
84                 System.err.println("Couldn't parse the init parameter date: " +
85                     ex.substring(0, ISO8601_LEN));
86                 e.printStackTrace();
87             }
88         }
89     }
90
91
92     /**
93     * Gets an ISO 8601 string representation of today's date.
94     */
95 }
```

```

95     * @return The date string
96     */
97     public static String getTodayString()
98     {
99         Calendar rightNow = Calendar.getInstance();
100        StringBuffer today = new StringBuffer();
101        today.append(rightNow.get(Calendar.YEAR));
102        today.append("-");
103        if (rightNow.get(Calendar.MONTH)+ 1 < 10)
104        {
105            today.append("0");
106        }
107        today.append(rightNow.get(Calendar.MONTH) + 1);
108        today.append("-");
109        if (rightNow.get(Calendar.DATE) < 10)
110        {
111            today.append("0");
112        }
113        today.append(rightNow.get(Calendar.DATE));
114
115        return today.toString();
116    }
117
118    /**
119     * Indicates if a game is scheduled for today.
120     */
121    public boolean isGameScheduledToday()
122    {
123        {
124            if (isRegularGameToday() && !isGameCancelledToday())
125            {
126                return true;
127            }
128
129            if (isExceptionGameToday())
130            {
131                return true;
132            }
133
134            return false;
135        }
136
137    /**
138     * Indicates if there is a regularly scheduled game today.
139     */
140    private boolean isRegularGameToday()
141    {
142        {
143            Calendar rightNow = Calendar.getInstance();
144            int day = rightNow.get(Calendar.DAY_OF_WEEK);
145
146            String dayStr = null;
147
148            switch (day)
149            {
150                case Calendar.SUNDAY:
151                    dayStr = "sun";
152                    break;
153
154                case Calendar.MONDAY:
155                    dayStr = "mon";
156                    break;
157
158                case Calendar.TUESDAY:
159                    dayStr = "tue";
160                    break;
161
162                case Calendar.WEDNESDAY:
163                    dayStr = "wed";
164                    break;
165
166                case Calendar.THURSDAY:
167                    dayStr = "thu";
168                    break;
169
170                case Calendar.FRIDAY:
171                    dayStr = "fri";
172                    break;
173
174                case Calendar.SATURDAY:
175                    dayStr = "sat";
176                    break;
177
178                default:
179                    System.err.println("Somebody has invented a new day of the week: " + day);
180            }
181
182            return -1 != m_schedule.indexOf(dayStr);
183        }
184
185    /**
186     * Indicates if an exception game is set for today.
187     */
188    private boolean isExceptionGameToday()
189    {
190        {
191            return false;

```

```

192     }
193
194
195     /**
196      * Indicates if a regular game is cancelled by exception today.
197      */
198     private boolean isGameCancelledToday()
199     {
200         return false;
201     }
202 }
203
204
205 /**
206  * Represents an exception to the normal schedule.
207  */
208 class ScheduleException
209 {
210     private Date m_date;
211     private String m_comment;
212
213     public ScheduleException(Date date, String comment)
214     {
215         m_date = date;
216         m_comment = comment;
217     }
218
219     public Date getDate()
220     {
221         return m_date;
222     }
223
224
225     public String getComment()
226     {
227         return m_comment;
228     }
229 }
230 }
231
232

```