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

```java
/*      $Workfile: $
    $Modtime: $
    $Author: $

    Originally created on July 25, 2002 by Eric Smith.

    Copyright (c) 2002, Eric Smith. All rights reserved.
*/

package ericsmith.hoops;



public class ConfigurationException extends Exception
{
    public ConfigurationException(String message)
    {
        super(message);
    }
}
```

```java
/*
 * $Workfile: $
 * $Modtime: $
 * $Author: $
 * Originally created on July 24, 2002 by Eric Smith.
 * Copyright (c) 2002, Eric Smith. All rights reserved.
 */
package ericsmith.hoops;

import ericsmith.util.FileUtils;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.util.Calendar;
import java.util.Iterator;
import java.util.List;
import org.jdom.Document;
import org.jdom.Element;
import org.jdom.JDOMException;
import org.jdom.input.SAXBuilder;
import org.jdom.output.XMLOutputter;
import org.jdom.transform.JDOMSource;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.stream.StreamSource;
import javax.xml.transform.stream.StreamResult;


/**
 * Represents the status for a game.
 *
 * @author Eric
 * @created July 25, 2002
 */
public class GameStatus
{
    private final static String GAME_TEMPLATE = "game-template.xml";

    private Document m_doc;
    private File m_gameFile;


    /**
     * Constructor for the GameStatus object. Clients must use the getGame
     * factory method to create instances.
     */
    private GameStatus() { }


    /**
     * Constructor for the GameStatus object. Clients must use the getGame
     * factory method to create instances.
     *
     * @param data XML file containing game data.
     * @param isNew indicates if this is a new game, or existing.
     * @exception ConfigurationException If there is a problem with the game
     *      template file.
     */
    private GameStatus(File data, boolean isNew, String date) throws ConfigurationException
    {
        m_gameFile = data;

        SAXBuilder builder = new SAXBuilder();

        try
        {
            m_doc = builder.build(new FileInputStream(data));

            if (isNew)
            {
                addDate(date);
            }
        }
        catch (FileNotFoundException ex)
        {
            throw new ConfigurationException(ex.getMessage());
        }
        catch (JDOMException ex)
        {
            throw new ConfigurationException(ex.getMessage());
        }
    }


    /**
     * Adds the given date to the current game document.
     */
    private void addDate(String dateStr)
    {
        Element game = m_doc.getRootElement();
        Element details = game.getChild("details");
        Element date = new Element("date");
        date.setText(dateStr);
        details.addContent(date);
    }


    /**
     * Factory method that gets the game status for the given day's game. If
     * the game status doesn't exist, it is created.
     *
     * @param date the day to retrieve game data for.
     * @param configDir directory to retrieve configuration info from.
     * @param dataDir directory to store and retrieve data files from.
     * @return The game
     * @exception ConfigurationException If there is a problem with the game
     *      template file.
     */
    public static GameStatus getGame(String date, String configDir, String dataDir) throws ConfigurationExcept «
ion
    {
        String gameFileName = date + ".xml";
        File gameFile = new File(dataDir + gameFileName);

        try
        {
            if (gameFile.exists())
            {
                return new GameStatus(gameFile, false, date);
            }
            else
            {
                FileUtils.copy(configDir + GAME_TEMPLATE, dataDir + gameFileName);
                return new GameStatus(gameFile, true, date);
            }
        }
        catch (IOException ex)
        {
            throw new ConfigurationException(ex.getMessage());
        }
    }


    /**
     * Writes the game status as XML.
     *
     * @param out destination stream.
     * @exception IOException If there is a problem writing to the supplied stream.
     */
    public void serialize(OutputStream out) throws IOException
    {
        XMLOutputter outputter = new XMLOutputter();
        outputter.output(m_doc, out);
    }


    /**
     * Writes the game status as the result of a transformation.
     *
     * @param name name of the currently logged-in player.
     * @param style path to the style sheet to transform with.
     * @param out destination stream.
     * @exception IOException If there is a problem writing to the supplied stream.
     */
    public void transform(String name, String style, OutputStream out) throws IOException, TransformerExceptio «
n
    {
        Transformer transformer = TransformerFactory.newInstance().newTransformer(new StreamSource(style));
        transformer.setParameter("current-player", name);
        transformer.transform(new JDOMSource(m_doc), new StreamResult(out));
    }


    /**
     * Saves the game data to a file to keep it persisent.
     *
     * @throws IOException If there is a problem writing the file.
     */
    public void save() throws IOException
    {
        XMLOutputter outputter = new XMLOutputter();
        FileOutputStream out = null;
        try
        {
            out = new FileOutputStream(m_gameFile);
            outputter.output(m_doc, out);
        }
        finally
        {
            if (out != null)
            {
                out.close();
            }
        }
    }

    /**
     * Sets the game status for a particular player.
     *
     * @param name the player's name
     * @param status the player's status.
     * @param comment comment from the player.
     */
    public void setPlayerStatus(String name, String status, String comment)
    {
        if (status == null)
        {
            status = "";
        }

        if (comment == null)
        {
            comment = "";
        }

        Element game = m_doc.getRootElement();
        Element players = game.getChild("players");

        Element player = getPlayerElement(players, name);

        Element playerStatus = player.getChild("status");
        playerStatus.setText(status);

        Element playerComment = player.getChild("comment");
        playerComment.setText(comment);
    }


    /**
     * Gets a player element by name. If the element doesn't exist, it is created
     * and added to the players element.
     *
     * @param players the players element containing the player.
     * @param name the name of the player to retrieve.
     * @return the player element.
     */
    private Element getPlayerElement(Element players, String name)
    {
        List playerList = players.getChildren();
        Iterator iter = playerList.iterator();
        while (iter.hasNext())
        {
            Element player = (Element) iter.next();
            if (player.getChildTextNormalize("name").equals(name))
            {
                return player;
            }
        }

        Element player = new Element("player");

        Element playerName = new Element("name");
        playerName.setText(name);
        player.addContent(playerName);

        Element playerStatus = new Element("status");
        player.addContent(playerStatus);

        Element playerComment = new Element("comment");
        player.addContent(playerComment);

        players.addContent(player);

        return player;
    }
}
```

```java
/*     $Workfile: $
    $Modtime: $
    $Author: $

    Originally created on July 25, 2002 by Eric Smith.

    Copyright (c) 2002, Eric Smith. All rights reserved.
*/

package ericsmith.hoops;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.Enumeration;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import javax.servlet.ServletConfig;
import javax.xml.transform.TransformerException;


/**
 * @author Eric
 * @created July 13, 2002
 */
public final class HoopsServlet extends HttpServlet
{
    private final static String DATA_DIR = "WEB-INF\\data\\";
    private final static String CONFIG_DIR = "WEB-INF\\config\\";
    private final static String STYLE_SHEET = "status.xsl";

    private ScheduleManager scheduleManager;
    private NotificationManager m_notificationManager;


    /**
     * Initialize the servlet before any requests come through.
     */
    public void init(ServletConfig config) throws ServletException
    {
        super.init(config);
        scheduleManager = new ScheduleManager(config);
        m_notificationManager = new NotificationManager(
                getServletContext().getRealPath("/") + CONFIG_DIR,
                getServletContext().getRealPath("/") + DATA_DIR);
    }


    /**
     * Respond to a GET request for the content produced by this servlet.
     *
     * @param request The servlet request we are processing
     * @param response The servlet response we are producing
     * @exception IOException if an input/output error occurs
     * @exception ServletException if a servlet error occurs
     */
    public void doGet(HttpServletRequest request, HttpServletResponse response)
             throws IOException, ServletException
    {

        response.setContentType("text/html");

        try
        {
            if (!scheduleManager.isGameScheduledToday())
             {
                // Send 'no game today' page.
                response.sendRedirect("no-game.html");
                return;
            }

            String playerName = getPlayer(request);
            if (playerName == null)
            {
                // Send login page.
                response.sendRedirect("login.html");
                return;
            }

            GameStatus game = GameStatus.getGame(scheduleManager.getTodayString(),
                getServletContext().getRealPath("/") + CONFIG_DIR,
                getServletContext().getRealPath("/") + DATA_DIR);

            String status = request.getParameter("ps");
            String comment = request.getParameter("pc");
            if (status != null || comment != null)
            {
                game.setPlayerStatus(playerName, status, comment);
            }

            if (request.getParameter("raw") != null)
            {
                 // Return the XML game data to the response stream.
                 response.setContentType("text/xml");
                 game.serialize(response.getOutputStream());
             }
             else
             {
                 game.transform(playerName, getServletContext().getRealPath("/") + STYLE_SHEET, response.getOut «
putStream());
             }

             game.save();
         }
         catch (ConfigurationException ex)
         {
             throw new ServletException(ex.getMessage());
         }
         catch (TransformerException ex)
         {
             throw new ServletException("There was a problem transforming the output with " + STYLE_SHEET + ".\ «
n" + ex.getMessage());
         }

     }


     /**
      * Gets the name of the logged-in player, if known.
      * @return The player's name, or null if it is not known.
      */
     private String getPlayer(HttpServletRequest request)
     {
         // Honor logout request first.
         if (request.getParameter("logout") != null && !request.getSession().isNew())
         {
             request.getSession().invalidate();
         }

         HttpSession session = request.getSession();
         String playerName = (String)session.getAttribute("name");
         if (playerName == null || playerName.equals(""))
         {
             playerName = request.getParameter("pn");
             if (playerName != null && !playerName.equals(""))
             {
                 session.setAttribute("name", playerName);
             }
         }

         if (playerName != null && playerName.equals(""))
         {
             playerName = null;
         }

         return playerName;
     }

 }
```

```
/*     $Workfile: $
    $Modtime: $
    $Author: $

    Originally created on September 21, 2002 by Eric Smith.

    Copyright (c) 2002, Eric Smith. All rights reserved.
*/

package ericsmith.hoops;


public class MailUtils
{
    /** SMTP mail server */
    private String m_mailServer;


    /**
     * Sends an HTML e-mail message.
     *
     * @param content The HTML content of the mail message.
     */
    public void sendMessageHTML(String content)
    {
        Properties mailProps = System.getProperties();
        mailProps.put("mail.smtp.host", m_mailServer);
        Session session = Session.getInstance(mailProps, null);
        Transport tr = session.getTransport("smtp");
        tr.connect(m_mailServer, smtpUserName, smtpPassword);
        MimeMessage message = new MimeMessage(session);
        message.setFrom(new InternetAddress(from));
        message.setRecipients(Message.RecipientType.TO, InternetAddress.parse(to));
        message.setSubject(subject);

        message.setContent(content, "text/html");

        if (useAccount)
        {
            tr.send(message);
            tr.close();
        }
        else
        {
            Transport.send(message);
        }
    }


    /**
     * Sends a plain text e-mail message.
     *
     * @param content The text content of the mail message.
     */
    public void sendMessageText(String content)
    {
    }



}
```

```java
/*     $Workfile: $
    $Modtime: $
    $Author: $

    Originally created on September 20, 2002 by Eric Smith.

    Copyright (c) 2002, Eric Smith. All rights reserved.
*/

package ericsmith.hoops;

import java.util.Date;
import java.util.Timer;
import java.util.TimerTask;


/**
 * Class to manage e-mail notifications to players.
 */
public class NotificationManager
{
    /** Number of milliseconds in a day. */
    private static final int ONE_DAY = 10000; //86400000

    /** Directory where configuration data is stored. **/
    private String m_configDir;

    /** Directory where application data is stored. **/
    private String m_dataDir;

    /** A timer for seding out e-mail invitations to a day's game. */
    private Timer m_inviteTimer = new Timer(true);


    /**
     * Constructor.
     */
    public NotificationManager(String configDir, String dataDir)
    {
        m_configDir = configDir;
        m_dataDir = dataDir;

        m_inviteTimer.scheduleAtFixedRate(new InviteTask(), new Date(), ONE_DAY);
    }


    /**
     * Sends an e-mail invitation to log in to the system.
     */
    public void sendInvitation()
    {
    }
}



/**
 * TimerTask that sends out game invitations.
 */
class InviteTask extends TimerTask
{
    public void run()
    {
        System.out.println("Do yo want to play basketball?");
    }
}
```

```java
/*     $Workfile: $
    $Modtime: $
    $Author: $

    Originally created on September 5, 2002 by Eric Smith.

    Copyright (c) 2002, Eric Smith. All rights reserved.
*/

package ericsmith.hoops;


import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.List;
import java.text.SimpleDateFormat;
import java.text.ParseException;
import java.util.StringTokenizer;
import javax.servlet.ServletConfig;



/**
 * Abstracts the game scheduling.
 */
public class ScheduleManager
{
    /** The length of an ISO 8601 date */
    private static final int ISO8601_LEN = 10;

    private String m_schedule;
    private ArrayList m_additions = new ArrayList();
    private ArrayList m_cancellations = new ArrayList();


    /**
     * Constructor.
     *
     * @param config The Servlet configuration from which to read schedule data.
     */
    public ScheduleManager(ServletConfig config)
    {
        m_schedule = config.getInitParameter("schedule").toLowerCase();
        String additions = config.getInitParameter("schedule-additions");
        if (additions != null)
        {
            parseExceptions(additions, m_additions);
        }

        String cancellations = config.getInitParameter("schedule-cancellations");
        if (cancellations != null)
        {
            parseExceptions(cancellations, m_cancellations);
        }
    }


    /**
     * Parses an exception list and adds the exceptions to a list.
     *
     * @param s The exception list to parse.
     * @param list The list to add exceptions to.
     */
    private void parseExceptions(String s, List list)
    {
        SimpleDateFormat dateFormat = new SimpleDateFormat();

        StringTokenizer st = new StringTokenizer(s, "|");
        while (st.hasMoreElements())
        {
            String ex = st.nextToken();

            try
            {
                ScheduleException exception =
                    new ScheduleException(dateFormat.parse(ex.substring(0, ISO8601_LEN)),
                    ex.substring(ISO8601_LEN));

                list.add(exception);
            }
            catch (ParseException e)
            {
                System.err.println("Couldn't parse the init parameter date: " +
                    ex.substring(0, ISO8601_LEN));
                e.printStackTrace();
            }
        }
    }


    /**
     * Gets an ISO 8601 string representation of today's date.
     *
     * @return The date string
     */
    public static String getTodayString()
    {
        Calendar rightNow = Calendar.getInstance();
        StringBuffer today = new StringBuffer();
        today.append(rightNow.get(Calendar.YEAR));
        today.append("-");
        if (rightNow.get(Calendar.MONTH)+ 1 < 10)
        {
            today.append("0");
        }
        today.append(rightNow.get(Calendar.MONTH) + 1);
        today.append("-");
        if (rightNow.get(Calendar.DATE) < 10)
        {
            today.append("0");
        }
        today.append(rightNow.get(Calendar.DATE));

        return today.toString();
    }


    /**
     * Indicates if a game is scheduled for today.
     */
    public boolean isGameScheduledToday()
    {
        if (isRegularGameToday() && !isGameCancelledToday())
        {
            return true;
        }

        if (isExceptionGameToday())
        {
            return true;
        }

        return false;
    }


    /**
     * Indicates if there is a regularly scheduled game today.
     */
    private boolean isRegularGameToday()
    {
        Calendar rightNow = Calendar.getInstance();
        int day = rightNow.get(Calendar.DAY_OF_WEEK);

        String dayStr = null;

        switch (day)
        {
            case Calendar.SUNDAY:
            dayStr = "sun";
            break;

            case Calendar.MONDAY:
            dayStr = "mon";
            break;

            case Calendar.TUESDAY:
            dayStr = "tue";
            break;

            case Calendar.WEDNESDAY:
            dayStr = "wed";
            break;

            case Calendar.THURSDAY:
            dayStr = "thu";
            break;

            case Calendar.FRIDAY:
            dayStr = "fri";
            break;

            case Calendar.SATURDAY:
            dayStr = "sat";
            break;

            default:
                System.err.println("Somebody has invented a new day of the week: " + day);
        }

        return -1 != m_schedule.indexOf(dayStr);
    }


    /**
     * Indicates if an exception game is set for today.
     */
    private boolean isExceptionGameToday()
    {
        return false;
```

```java
    }


    /**
     * Indicates if a regular game is cancelled by exception today.
     */
    private boolean isGameCancelledToday()
    {
        return false;
    }
}


/**
 * Represents an exception to the normal schedule.
 */
class ScheduleException
{
    private Date m_date;
    private String m_comment;

    public ScheduleException(Date date, String comment)
    {
        m_date = date;
        m_comment = comment;
    }


    public Date getDate()
    {
        return m_date;
    }


    public String getComment()
    {
        return m_comment;
    }
}
```