```java
package lib;

/*******************************************************************************
  * ModuleManager.java
  * Created on March 26, 2013 by Matthew A. Crist.
  *
  * This class will manage to loading of module information from the
  * modules.xml file that is stored in the ./config directory.
  *
  * CHANGE LOG:
  * -----------------------------------------------------------------------------
  * 2013-03-26  -   Initial conception of this file.
  *
  ******************************************************************************/

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;

public class ModuleManager extends JDialog {
    public ArrayList<Module> modules = new ArrayList<Module>();

    private JDialog    _addDialog;
    private JDialog    _modDialog;
    private JTextField _fileInput;
    private JTextField _nameInput;
    private JTextField _portInput;
    private JButton    _removeModule;
    private JButton    _modifyModule;
    private JButton    _saveModules;
    private JList      moduleList;
    private JPanel     _addLayout;
    private JPanel     _manageLayout;

    private Module _temp;

    /**
     * Default, unargumented constructor for this class.
     */
    public ModuleManager() {
        modules = loadModules("config/modules.xml");
    }   // end default, unargumented constructor

    /**
     * Acquires the dialog that will allow a user to manage the current modules
     * that are registered with the server.
     */
    public void getManageModuleDialog() {
        _modDialog    = new JDialog();
        _manageLayout = new JPanel(null);

        DefaultListModel<String> listModel = new DefaultListModel<String>();
        moduleList = new JList(listModel);
        moduleList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        JScrollPane scroller = new JScrollPane(moduleList);

        // Add all the modules to the list model
        for(int i = 0; i < this.modules.size(); i++) {
            listModel.addElement(this.modules.get(i).getName());
        }   // end for loop

        JButton _addButton = new JButton("Add", new ImageIcon("lib/brick_add.png", "Add Module"));
        _removeModule = new JButton("Remove", new ImageIcon("lib/brick_delete.png", "Remove Module"));
    _modifyModule = new JButton("Modify", new ImageIcon("lib/brick_edit.png", "Modify Module"));
    _saveModules  = new JButton("Save", new ImageIcon("lib/disk.png", "Save Modules"));
    JButton _cancelButton = new JButton("Cancel", new ImageIcon("lib/cancel.png", "Cancel Action"));

    _removeModule.setEnabled(false);
    _modifyModule.setEnabled(false);

    // List Selection for button activation
    moduleList.addListSelectionListener(new ListSelectionListener() {
        public void valueChanged(ListSelectionEvent lse) {
            if(lse.getValueIsAdjusting() == false) {
                if(moduleList.getSelectedIndex() == -1) {
                    _removeModule.setEnabled(false);
                    _modifyModule.setEnabled(false);
                } else {
                    _removeModule.setEnabled(true);
                    _modifyModule.setEnabled(true);
                }   // end if-else
            }   // end if
        }   // end method valueChanged
    });

    // Add button action listener
    _addButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent ae) {
            _modDialog.setVisible(false);
            getAddModuleDialog();
        }   // end method actionPerformed
    });

    _saveModules.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent ae) {
            boolean status = storeModules("config/modules.xml");

            if(status) {
                JOptionPane.showMessageDialog(((JButton)ae.getSource()).getRootPane(), "Modules updated ↙
 successfully!");
                _modDialog.dispose();
            } else {
                JOptionPane.showMessageDialog(((JButton)ae.getSource()).getRootPane(), "Unable to      ↙
update modules.  Please see console output.", "Module Management Error", JOptionPane.ERROR_MESSAGE);
                _modDialog.dispose();
            }   // end if-else
        }   // end method actionPerformed
    });

    _cancelButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent ae) {
            _modDialog.dispose();
        }   // end method actionPerformed
    });

    _modifyModule.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent ae) {
            Module _selected = getModule((String)moduleList.getSelectedValue());
            getAddModuleDialog(_selected.getName(), _selected.getInvocation(), _selected.getPort());
            _modDialog.setVisible(false);
        }   // end method actionPerformed
    });

    _manageLayout.setBorder(BorderFactory.createTitledBorder("Module Management"));

    scroller.setBounds(20, 20, 200, 300);
    _addButton.setBounds(230, 20, 100, 20);
    _modifyModule.setBounds(230, 45, 100, 20);
    _removeModule.setBounds(230, 70, 100, 20);
    _saveModules.setBounds(230, 300, 100, 20);
    _cancelButton.setBounds(230, 275, 100, 20);
```

```
        _manageLayout.add(scroller);
        _manageLayout.add(_addButton);
        _manageLayout.add(_modifyModule);
        _manageLayout.add(_removeModule);
        _manageLayout.add(_saveModules);
        _manageLayout.add(_cancelButton);

        _modDialog.add(_manageLayout);

        int width  = Toolkit.getDefaultToolkit().getScreenSize().width;
        int height = Toolkit.getDefaultToolkit().getScreenSize().height;
        _modDialog.setBounds(width/2-185, height/2-190, 370, 380);
        _modDialog.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        _modDialog.setVisible(true);
    }   // end method getManageModuleDialog

    /**
     * Overloaded getAddModuleDialog that will get the "ADD" action to a module.
     */
    public void getAddModuleDialog() {
        getAddModuleDialog(null, null, null);
    }   // end method getAddModuleDialog

    /**
      * Acquires the dialog that will allow a user to register a module with the
      * server for monitoring.  EDIT by default for parameters passed.
      */
    public void getAddModuleDialog(String t_name, String t_invoke, String t_port) {
        final String _name = t_name;
        final String _invoke = t_invoke;
        final String _port = t_port;

        _addLayout = new JPanel(null);
        _addDialog = new JDialog();

        JLabel _nameLabel   = new JLabel("Module Name:");
        JLabel _invokeLabel = new JLabel("Script to Invoke:");
        JLabel _portLabel   = new JLabel("Listening Port:");

        _nameInput  = new JTextField();
        _fileInput  = new JTextField();
        _portInput  = new JTextField();

        // Determine if we are editing or it is a new entry
        if((_name != null) && (_invoke != null) && (_port != null)) {
            _nameInput.setText(_name);
            _fileInput.setText(_invoke);
            _portInput.setText(_port);

            // Temporarily store the old name to ensure it's removal if edited.
            _temp = getModule(_name);
            removeModule(_name);
        }   // end if

        JButton _fileSelect       = new JButton("Select...");
        JButton _saveModuleButton = new JButton("Save");
        JButton _cancelButton     = new JButton("Cancel");

        _fileSelect.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent ae) {
                JFileChooser chooser = new JFileChooser(".");
                int val = chooser.showOpenDialog(_addDialog);

                if(val == JFileChooser.APPROVE_OPTION) {
                    String filePath      = chooser.getSelectedFile().getPath();
                    // Kill the absolute package resolution
                    String modulePackage = filePath.substring(filePath.lastIndexOf("server"), filePath.    ↙
    length()-1);
                    // Replace the file separators with . for the package name resolution
                    modulePackage = modulePackage.replace(System.getProperty("file.separator"), ".");
                    // Trim off server. portion of string
                    modulePackage = modulePackage.substring(modulePackage.indexOf(".")+1, modulePackage.    ↙
    length()-1);
                    // Trim off the file extension.
                    modulePackage = modulePackage.substring(0, modulePackage.lastIndexOf("."));
                    _fileInput.setText(modulePackage);
                }   // end if
            }   // end method actionPerformed
        });

        // The save action against the new module.
        _saveModuleButton.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent ae)
            {
                Module module = new Module();
                module.setName(_nameInput.getText());
                module.setInvocation(_fileInput.getText());
                module.setPort(_portInput.getText());

                // Remove the old module if declared and reset the name handler
                if(_temp != null) {
                    _temp = null;
                }   // end if

                // Add the module and dispose of the window
                addModule(module);
                _addDialog.dispose();

                getManageModuleDialog();
            }   // end method actionPerformed
        });

        /**
         * ActionListener for the cancel button that will discard any changes that
         * occured to the module, and dispose of the frame.
         */
        _cancelButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent ae) {
                if(_temp != null) {
                    addModule(_temp);
                    _temp = null;
                }
                _addDialog.dispose();
                getManageModuleDialog();
            }   // end method actionPerformed
        });

        _nameLabel.setBounds(20, 20, 100, 20);
        _invokeLabel.setBounds(20, 45, 100, 20);
        _portLabel.setBounds(20, 70, 100, 20);

        _nameInput.setBounds(130, 20, 150, 20);
        _fileInput.setBounds(130, 45, 150, 20);
        _portInput.setBounds(130, 70, 150, 20);

        _fileSelect.setBounds(290, 45, 100, 20);

        _addLayout.add(_nameLabel);
        _addLayout.add(_nameInput);
        _addLayout.add(_invokeLabel);
        _addLayout.add(_fileInput);
        _addLayout.add(_fileSelect);
        _addLayout.add(_portLabel);
        _addLayout.add(_portInput);
        JSeparator sep = new JSeparator();
        sep.setBounds(20, 95, 380, 2);
        _addLayout.add(sep);

        _cancelButton.setBounds(190, 105, 100, 20);
        _addLayout.add(_cancelButton);
        _saveModuleButton.setBounds(300, 105, 100, 20);
        _addLayout.add(_saveModuleButton);

        _addLayout.setBorder(BorderFactory.createTitledBorder("Register Module"));

        _addDialog.add(_addLayout);
        _addDialog.setBounds(0,0, 420, 170);
        _addDialog.setResizable(false);
        _addDialog.setVisible(true);
        _addDialog.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    }   // end method getAddModuleDialog

    /**
      * Adds a module to the modules list.
     */
    public boolean addModule(Module module) {
        boolean ok = true;

        for(int i = 0; i < this.modules.size(); i++) {
            if(this.modules.get(i).getPort().equalsIgnoreCase(module.getPort())) {
                ok = false;
            }   // end if
        }   // end for loop

        // Attempt to add the module.
        if(ok) {
            this.modules.add(module);
            this.storeModules("config/modules.xml");

            return true;
        } else {
            JOptionPane.showMessageDialog(this, "Unable to add module.  There exists a module using the     ↙
    same port as the one you are attempting to register.", "Error Registering Module", JOptionPane.      ↙
    ERROR_MESSAGE);
            return false;
        }   // end if-else
    }   // end method addModule

    /**
     * Removes a module from the modules list.
      */
    public boolean removeModule(String name) {
        boolean removed = false;

        for(int i = 0; i < this.modules.size(); i++) {
            if(this.modules.get(i).getName().toLowerCase().equals(name.toLowerCase())) {
                this.modules.remove(i);
                removed = true;
            }   // end if
        }   // end for loop

        return removed;
    }   // end method removeModule

    /**
     * Acquires a module by its name.
     */
    public Module getModule(String name) {
        for(int i = 0; i < this.modules.size(); i++) {
            if(this.modules.get(i).getName().toLowerCase().equals(name.toLowerCase())) {
                return this.modules.get(i);
            }   // end if
        }   // end for loop

        return null;
    }   // end method getModule

    /**
     * Loads the modules from a persistent XML storage file.
     */
    public ArrayList<Module> loadModules(String configFile)
    {
        ArrayList<Module> modules = new ArrayList<Module>();

        try
        {
            File moduleFile = new File(configFile);
            DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
            DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
            Document doc = dBuilder.parse(moduleFile);

            doc.getDocumentElement().normalize();

            NodeList nodes = doc.getElementsByTagName("module");

            for(int i = 0; i < nodes.getLength(); i++)
            {
                Node node = nodes.item(i);

                if(node.getNodeType() == Node.ELEMENT_NODE)
                {
                    Element element = (Element)node;
                    Module module = new Module();
                    module.setName(element.getElementsByTagName("name").item(0).getTextContent());
                    module.setInvocation(element.getElementsByTagName("invoke").item(0).getTextContent());
                    module.setPort(element.getElementsByTagName("port").item(0).getTextContent());

                    modules.add(module);
                }
            }
        }   // end try
        catch(Exception e)
        {
            e.printStackTrace();
        }   // end catch

        return modules;
    }   // end method loadModules

    /**
     * Writes the modules out to an XML file for storage.
     */
    public boolean storeModules(String configFile)
    {
        // Header information for the XML file.
        String xml = "<?xml version='1.0'?>\n"                                + 
                     "<!DOCTYPE modules SYSTEM '../dtd/module.dtd'>\n" +
                     "<modules>\n";

        // Cycle through all modules and compile the XML for output.
        for(int i = 0; i < this.modules.size(); i++)
        {
            xml += "\t<module>\n"                                                  +
                     "\t\t<name>" + this.modules.get(i).getName() + "</name>\n"            +
                "\t\t<invoke>" + this.modules.get(i).getInvocation() + "</invoke>\n"  +
                "\t\t<port>" + this.modules.get(i).getPort() + "</port>\n" +
                     "\t</module>\n";
        }   // end for loop
        xml += "</modules>";

        try
        {
            FileWriter writer = new FileWriter(configFile);
            writer.write(xml);
            writer.close();
        }   // end try
        catch(IOException ioe)
        {
            ioe.printStackTrace();
            return false;
        }   // end catch

        return true;
    }   // end method storeModules

    /**
     * Acquires the set of modules that are currently stored in the module manager.
     */
    public ArrayList<Module> getModules()
    {
        return this.modules;
    }   // end method getModules
}   // end class ModuleManager
```