Comments about Assignment 1

Smalltalkers ................................................................. 3
Java ................................................................. 4
  Duh Comments ....................................................... 4
  Name & Comments ................................................ 7
  Formatting .......................................................... 12
  Line Wrap ......................................................... 14
  Naming Convention ............................................. 17
  Nested Classes ................................................... 18
  Exceptions & Users ............................................... 20
  Repeated Code ................................................... 21
  Partial Java Solution .......................................... 24
    RemoteException ............................................... 24
    FacultyDataProxy .............................................. 25
    Callback .......................................................... 30
    CallbackCommand ............................................ 31
    DepartmentListCommand .................................... 33
    FacultyListCommand ......................................... 34
    FacultyDataCommand ......................................... 35
    MockListener ................................................ 37

References

Student papers

Refactoring: Improving the Design of Existing Code, Fowler, Addison-Wesley, 1999
<table>
<thead>
<tr>
<th></th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>A working program</td>
<td>8</td>
</tr>
<tr>
<td>Comments</td>
<td>4</td>
</tr>
<tr>
<td>Code Quality</td>
<td>8</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>20</strong></td>
</tr>
</tbody>
</table>
Smalltalkers

Run Code Critic on your code

Use the formatter to format your code
// Packages imported
import java.awt.*;
import javax.swing.*;

public class MyGUIClient {  //The GUI class
    // contents
    JFrame facultyName;
    JButton exitButton;  //button used to exit program
    JButton cancelButton;  //button used to cancel request
    JComboBox departmentList;  //List of departments
    //Constructor
    public MyGUIClient() {
        initGUI();  // initialize the gui
    }

    private void initGUI() {  //initialize the GUI
        // Create listener
        FacultyListListener aListener = new FacultyListListener();

        // Create toolbar
        toolbar = new JToolBar();
    }

    //Method: construct
    //Operation: Perform the xmlrpc query
    public Object construct() {
        parameter = new Vector();  //empty parameter
        name = (Vector) data.get("name");  //extracting the data in vector
import java.awt.*;
import javax.swing.*;

public class GUIClient {
    JFrame facultyName;
    JButton exitProgram;
    JButton cancelRequest;
    JComboBox departmentList;

    public MyGUIClient() {
        initializeGUI();
    }

    private void initializeGUI() {
        FacultyListListener aListener = new FacultyListListener();

        toolbar = new JToolBar();
    }

    public Object performQuery() {
        parameter = new Vector(); // Why is this here?
        name = (Vector) data.get("name");
    }
}
Comments

Comments are good

Often comments are used as a deodorant to cover up the smell of bad code

“When you feel the need to write a comment, first try to refactor the code so that any comment becomes superfluous”

Fowler

Don’t repeat code in your comment
Name & Comments

Names of variables, classes and methods should be expressive

“Cancel request for department list
cancelButton[0] = new JButton(“cancel”);

verses

cancelDepartmentList = new JButton(“cancel”);
Variable Names

With a good name variables don’t need comments explaining their role

They may need comments explaining:

• Units

final static int BUTTON_HEIGHT = 50;             //in pixels

• Legal values

int requestType;          //legal range 0 to 5
Comments and Blocks

If

- You have long methods
- Comment blocks of code

Perhaps you should make each commented block a method
Using numbers in names rarely is a good idea

```java
JLabel label1;
JLabel label2;
```

verses

```java
JLabel officeNumber;
JLabel phoneNumber;
```

**My in names**

```java
public class MyGUI
```

The My does not add anything to the name
class p1

These names are not descriptive

They will not help the reader

public class Hmw1e
{
    public static void main(String[] args)
    {
        JFrame frame = new InfoListsFrame();
        frame.show();
    }
}

public class assignment1_version_3 {
class FacultyInfoSystemFrame extends JFrame 
    implements ActionListener, TreeSelectionListener, Runnable 
{

    private DefaultMutableTreeNode root;
    private DefaultTreeModel model;
    private JTree tree;
    private JTextArea textArea;
    private Thread doActionThread = null;
    private JMenu fileMenu;
    private JMenuItem exitMenuItem;
    private JMenuBar menuBar;
    private JMenuItem aboutMenuItem;
    private JMenu helpMenu;
    private JMenu modeMenu;
    private JMenuItem sleepMenuItem;
    private JMenu cancelMenu;
    private JMenuItem cancelcallMenuItem;

    FacultyInfoXmlRpcClient client = new FacultyInfoXmlRpcClient();
    Vector deptNameVect = new Vector();
    Vector deptIdVect = new Vector();
    Hashtable rtnValHash = new Hashtable();

    public FacultyInfoSystemFrame()
    {
        setTitle("Faculty Information System");
        setSize(1000, 600);
       setFont(new Font("Arial Black", 0, 22));
Don’t mix Tabs & Spaces

If you use tabs to indent
• Use tabs to indent on all lines

If you use spaces to indent
• Use spaces to indent on all lines

Never mix them

Some IDEs auto indent with:
• Tabs
• Spaces

Work it out with your IDE so you do not mix them together
public void setOfficeData(Vector officeData) {

    if (_facultyInformationView.getCancelButton().isEnabled()) {
        if (officeData.size() == 3) {
            boolean clearTable = false;
            int rows = _facultyInformationView.getOfficeDataTable().getRowCount();

            for (int i=0; i<rows; i++) {
                if ( ((String)_facultyInformationView.getOfficeDataTable().getValueAt( i, 0)).equals("")) {
                    _facultyInformationView.getOfficeDataTable().setValueAt(officeData.get(0), i, 0);
                    _facultyInformationView.getOfficeDataTable().setValueAt(officeData.get(1), i, 1);
                    _facultyInformationView.getOfficeDataTable().setValueAt(officeData.get(2), i, 2);
                    break;
                }
            }

            if (i == 9)
                clearTable = true;
        }
    }
}
Line Wrap – Work it out with your Printer

Find out how much can fit on one line

I add a comment with length of the page

I don’t all code to go past that length

//2345678901234567890123456789012345678901234567890
Keep Comments indented with Code

// Departments makes the XMLRPC connection and retrieves the list
// in a hash hashtable
Departments departmentsList = new Departments();
getDepartments(departmentsList.getDepartmentsHashtable());
departments.addActionListener(this);

JPanel deptPanel = new JPanel();
deptPanel.add(new JLabel("Department List: "));
deptPanel.add(departments);
getContentPane().add(deptPanel, "North");

// the other panels that handle the rest of functionality for
// the assignment
facultyPanel = new FacultyListPanel();
getContentPane().add(facultyPanel, "Center");

waitPanel = new SecondsToWaitPanel();
getContentPane().add(waitPanel, "South");
}

/** The argument, xmlRpcArgs is the hashtable that is returned
from the "departments" method on the server. The list of departments
is loaded into a combo box.
*/
public void getDepartments( Hashtable xmlRpcArgs )
{
deptList = xmlRpcArgs;
// "id" and "name" are the keys to the hashtable
idVec = (Vector)deptList.get("id");
nameVec = (Vector)deptList.get("name");

// setup the combo box to hold the department descriptions
for (int i = 0; i< nameVec.size(); i++)
    departments.addItem(nameVec.elementAt(i));
}
Naming Convention

All class names start with a capital letter

Variable names start with a lower case letter

Constants are all uppercase
Nested Classes

Some people used nested classes

Most who did

• used the outer class to define global variables
• Really had only one class

Avoid nested classes
Hard Coded Server URL

```java
try {
    xmlrpc = new XmlRpcClientLite
    ("http://rugby.sdsu.edu:8008/cs580");
}
```

Should clients have to be edited & recompiled

- each time the server address changes?
- For each site using the client?
Exceptions & Users

• Do users want to see a stack trace?
• Why do the same thing three times?

try
{
    xmlrpc = new XmlRpcClientLite
             ("http://rugby.sdsu.edu:8008/cs580");
}
catch (java.net.MalformedURLException badAddress)
{
    badAddress.printStackTrace( System.out);
}
catch (java.io.IOException connectionProblem)
{
    connectionProblem.printStackTrace( System.out);
}
catch (Exception serverProblem)
{
    serverProblem.printStackTrace( System.out);
}

try
{
    xmlrpc = new XmlRpcClientLite
             ("http://rugby.sdsu.edu:8008/cs580");
} //catches MalformedURLException, IOException
catch (Exception serverProblem)
{
    serverProblem.printStackTrace( System.out);
}
Repeated Code

public void departments ()
{
    String lastDepartmentId= null;
    Vector parameters = new Vector ();
    Hashtable departments =null;
    try
    {
        departments = (Hashtable) xmlrpc.execute("departments",parameters);
    }
    catch (java.net.MalformedURLException badAddress)
    {
        badAddress.printStackTrace( System.out);
    }
    catch (java.io.IOException connectionProblem)
    {
        connectionProblem.printStackTrace( System.out);
    }
    catch (Exception serverProblem)
    {
        serverProblem.printStackTrace( System.out);
    }
    if(!cancel)
        gui.setDepartments(departments);
}

public void departments (int timedelay)
{
    String lastDepartmentId= "pseudo";
    Vector parameters = new Vector ();
    parameters.add(new Integer(timedelay));
    Hashtable departments =null;
    try
    {
        departments = (Hashtable) xmlrpc.execute("delayDepartments",parameters);
    }
    catch (java.net.MalformedURLException badAddress)
    {
        badAddress.printStackTrace( System.out);
    }
catch (java.io.IOException connectionProblem)  
{  
    connectionProblem.printStackTrace( System.out);  
}  
catch (Exception serverProblem)  
{  
    serverProblem.printStackTrace( System.out);  
}  

if (!cancel)  
{  
    gui.setDepartments(departments);  
}  

public void facultyIn(String departmentId)  
{  
    Vector faculties=null;  
    Vector parameters = new Vector();  
    parameters.add(departmentId);  
    try  
    {  
        faculties = (Vector) xmlrpc.execute("facultyIn",parameters);  
    }  
    catch (java.net.MalformedURLException badAddress)  
    {  
        badAddress.printStackTrace( System.out);  
    }  
    catch (java.io.IOException connectionProblem)  
    {  
        connectionProblem.printStackTrace( System.out);  
    }  
    catch (Exception serverProblem)  
    {  
        serverProblem.printStackTrace( System.out);  
    }  
    if (!cancel)  
    {  
        gui.setFacultyIn(faculties);  
    }  
}
Once & Only Once

When you find yourself cutting & pasting code

Look for a way to do it in one location
Partial Java Solution
RemoteConnectionException

Used to wrap multiple exceptions into one exception

Uses new feature of JDK 1.4

```java
public class RemoteConnectionException extends Exception
{

    public RemoteConnectionException(Throwables causeOfException)
    {
        super(causeOfException);
    }
}
```
import org.apache.xmlrpc.*;
import java.util.*;
import java.net.MalformedURLException;

public class FacultyDataProxy
{
    private XmlRpcClient serverProxy;
    private Vector departmentNames;
    private Vector departmentIDs;

    public FacultyDataProxy(String serverUrl) throws MalformedURLException
    {
        serverProxy = new XmlRpcClientLite( serverUrl);
    }

    public Vector departmentNames()  throws RemoteConnectionException
    {
        if (departmentNames == null)
        {
            Hashtable departmentData;
            departmentData = (Hashtable) execute( "departments");
            departmentNames = (Vector) departmentData.get("name");
            departmentIDs = (Vector) departmentData.get("id");
        }
        return departmentNames;
    }

    public Vector departmentFaculty(String departmentName) throws RemoteConnectionException
    {
        int departmentIndex = departmentNames().indexOf( departmentName);
        String id = (String) departmentIDs.get( departmentIndex);
        return (Vector) execute( "facultyIn",id );
    }
}
public Vector facultyData(String name) throws RemoteException 
{
    return (Vector) execute( "officeDataFor", name );
}

private Object execute( String command, String parameter) 
    throws RemoteException 
{
    Vector parameters = new Vector();
    parameters.add( parameter);
    return execute(command, parameters);
}

private Object execute( String command) 
    throws RemoteException 
{
    return execute(command, new Vector());
}

private Object execute(String command, Vector parameters) 
    throws RemoteException 
{
    try
    {
        return serverProxy.execute(command, parameters);
    }
    catch (Exception executeProblem )
    {
        throw new RemoteException( executeProblem );
    }
}
}
Remarks about FacultyDataProxy

What needs to be commented?

• Server commands
• Data structures returned by server
• Format of URL
• Reason for caching department information
• Reason for hiding department Ids

Anything else?
Sample Tests for FacultyDataProxy

```java
import junit.framework.*;
import java.net.MalformedURLException;
import java.util.Vector;

public class TestFacultyDataProxy extends TestCase {
    public void testBadURL() {
        try {
            FacultyDataProxy server =
                new FacultyDataProxy("http://rugby.sdsu.edu:8008/cs580");
            fail("Should have thrown bad url exception");
        }
        catch (MalformedURLException badURL) {
            assertTrue("Exception should be thrown", true);
        }
        catch (Exception problem) {
            fail("No exception expected");
        }
    }
}
```
public void testDepartments()
{
    try
    {
        FacultyDataProxy server = server();
        Vector departmentList = server.departmentNames();
        assertTrue( departmentList.contains( "Computer Science"));
    }
    catch (Exception problem )
    {
        fail("No exception expected");
    }
}
Callback

Threads are going to callback to program to provide answers

Callback is the interface threads will use in calling back

```java
import java.util.Vector;

public interface Callback {
    public void commandError(Exception error);
    public void updateDepartmentList(Vector list);
    public void updateFacultyList(Vector list);
    public void updateFaculty(Vector facultyData);
}
```
CallbackCommand

Abstract class with algorithm for performing
• command and
• callback

Concrete subclasses implement command and callback details

public abstract class CallbackCommand extends Thread
{
  Callback commandListener;
  FacultyDataProxy commandPerformer;
  private int delayMilliseconds;

  protected abstract void performCommand()
      throws RemoteException;

  protected abstract void performCallback();

  public CallbackCommand(Callback listener,
      FacultyDataProxy performer,
      int delaySeconds)
  {
    commandPerformer = performer;
    commandListener = listener;
    delayMilliseconds = delaySeconds * 1000;
  }
}
CallbackCommand Continued

```java
public void run()
{
    try {
        sleep( delayMilliseconds);
        performCommand();
    }
    catch (InterruptedException haltCommand) {
        return;
    }
    catch (RemoteConnectionException error) {
        commandListener.commandError( error );
    }
    if (isInterrupted()) {
        return;
    }
    performCallback();
}
```
import java.util.Vector;

public class DepartmentListCommand extends CallbackCommand {
    private Vector departmentList;

    public DepartmentListCommand(Callback listener,
                                   FacultyDataProxy performer,
                                   int delaySeconds)
    {
        super( listener, performer, delaySeconds);
    }

    protected void performCommand() throws RemoteConnectionException
    {
        departmentList = commandPerformer.departmentNames();
    }

    protected void performCallback()
    {
        commandListener.updateDepartmentList( departmentList);
    }
}
FacultyListCommand

import java.util.Vector;

public class FacultyListCommand extends CallbackCommand
{
    private Vector facultyList;
    private String departmentName;

    public FacultyListCommand(Callback listener,
                               FacultyDataProxy performer,
                               int delaySeconds, String name)
    {
        super( listener, performer, delaySeconds);
        departmentName = name;
    }

    protected void performCommand() throws RemoteConnectionException
    {
        facultyList = commandPerformer.departmentFaculty(departmentName);
    }

    protected void performCallback()
    {
        commandListener.updateFacultyList( facultyList);
    }
}
FacultyDataCommand

import java.util.Vector;

public class FacultyDataCommand extends CallbackCommand
{
    private Vector facultyData;
    private String facultyName;

    public FacultyDataCommand(Callback listener,
                                FacultyDataProxy performer,
                                int delaySeconds,
                                String name)
    {
        super( listener, performer, delaySeconds);
        facultyName = name;
    }

    protected void performCommand() throws RemoteException
    {
        facultyData = commandPerformer.facultyData(facultyName);
    }

    protected void performCallback()
    {
        commandListener.updateFaculty( facultyData);
    }
}
public void testDepartmentListCommandInterrupt() {
    try {
        MockListener fakeGUI = new MockListener();
        FacultyDataProxy server = server();
        CallbackCommand command =
            new DepartmentListCommand(fakeGUI, server, 1);
        command.start();
        command.interrupt();
        assertFalse(fakeGUI.hasDepartmentList());
    } catch (Exception error) {
        fail("exception thrown");
    }
}
MockListener

Class used to help test Callback commands

```java
import java.util.Vector;

public class MockListener implements Callback {
    boolean exceptionOccured = false;
    Vector departmentList;
    Vector facultyList;
    Vector faculty;

    public void commandError(Exception error) {
        exceptionOccured = true;
    }

    public void updateDepartmentList(Vector list) {
        departmentList = list;
    }

    public void updateFacultyList(Vector list) {
        facultyList = list;
    }

    public void updateFaculty(Vector facultyData) {
        faculty = facultyData;
    }

    public boolean hasDepartmentList() {
        return !(departmentList == null);
    }
}
```