Homework 3– Distributing the University Directory application using Java RMI

Due: November 23, 2004

In this assignment, you will use Java RMI to distribute your application from Homework 2. You will use the "Person" class hierarchy and the *PersonList* class that implements the directory storing instances of Person.

The code is for *PersonList* is partially provided to you (see below); you only have to add the portion of code that binds the server implementation to the registry after instantiating it. To avoid changes to your *Person* class and all its subclasses, you will use an adapter, the *PersonAdapter*, that will "map" client calls to the correct methods on the *Person* class and its subclasses.

Thus you should have:

1- Two remote interfaces: one for *PersonList* and one for *PersonAdapter*. The *PersonList* interface will be the only one registered in the registry and the interface between the client and the remote server. The interface for *PersonAdapter* will enable the passing of *Person* objects by reference, instead of the passing by value.
2- For each class in the *Person* hierarchy, you only need a constructor (for the non-abstract classes), a "printDescription" method and one more method of your choice. The *printDescription* should keep the same functionality required for homework 2.
3- A *PersonListClient* class that gets a handle to the server and invokes methods on it (there are only two methods to invoke: *addperson* and *listAll*
4- Add any necessary code to demonstrate that the *Person* objects are actually being passed by reference.

For more details on the use of adapters, see the Chapter on Design Patterns in the book, as well as the link: http://www.javaworld.com/javaworld/jw-05-1999/jw-05-networked-p2.html

Adapters will enable the use of all the classes in the *Person* hierarchy without having to deal with the inheritance restrictions imposed by Java. In addition, there is no need for any changes on the classes in order to make them remotely accessible, e.g., they do not have to be rewritten as remote interfaces. Notice in the code below that the "directory" in *PersonListImpl* holds instances of *PersonAdapter* and not of *Person* anymore.

Partial code for *PersonList.java* and *PersonListImpl.java*

// PERSONLIST.JAVA

```
import java.rmi.*;
import java.util.Vector;
public interface PersonList extends Remote
```

```
{

      public void addPerson(PersonAdapter p) throws RemoteException;

      public String listAll() throws RemoteException;


 }
// PERSONLISTIMPL.java
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
import java.util.Vector;

public class PersonListImpl extends java.rmi.server.UnicastRemoteObject
                        implements PersonList{
// DATA
  private Vector theList;


 // METHODS
  // constructor: initializes the data
  public PersonListImpl()throws RemoteException
  {
      theList = new Vector();

  }
  // operations offered to clients:
  public void addPerson(PersonAdapter p) throws RemoteException
  {

      theList.addElement(p);       // the person is added to the list.

  }


  public String listAll()throws RemoteException
  {
      int i;
      String printstring = "";
       for (i=0; i<theList.size(); i++)
            printstring += ((PersonAdapter)theList.elementAt(i)).
pDescription();
      return printstring;
  }


public static void main(String args[])
  {
              // The Naming.rebind method may generate exceptions, thus
should be within the 'try' command
      try{

  // ADD YOUR CODE HERE TO INSTANTIATE A SERVER IMPLEMENTATION AND
  // REGISTER IT WITH THE REGISTRY.


          }
      // the 'catch' command will "treat" any exceptions tha may have
occured above
      catch(Exception e) {
            System.out.println("PersonList server main " + e.getMessage
());
        }
    }


 }
```

Deliverable

1- **Well documented code for all your classes**
2- **A page describing your design decisions**
3- **A short “user manual” describing how to use your program**

4- **No sample runs, please..**