Zebo Xiong - A 04907051

(1) Describe typical steps of actions (in the correct order) of a "WeatherService" RMI application session.

------------------ Server side

1) We need to registry the port for this RMI via command "rmiregistry 9999&"

2) We need to start WeatherServiceSearver with security control file via command "java -Djava.security.policy=policy1-file.txt WeatherServiceServer"

3) The WeatherService will contact the HTTP server and get weather information

4) Store the data from the HTTP server to WeatherBean.

5) Export the remote object using "UnicastRemoteObject".

6) The remote object will wait for client's call. At the same time, the "UnicastRemoteObject" is ready to throw an exception

7) Then we rebind the WeatherService stub (who will responsible for marshalling and unmarshalling) to the service name such as "*WeatherService*".

------------------- Client side

8) Then, we start the client with security policy file via command "java -Djava.security.policy=policy1-file.txt WeatherService"

9) The client gets the registry information from the port (such as 999) in server

10) The client invokes remote method getWeatherInformation of the interface WeatherService to obtain weather information.

11) The class uses a JList with a customerized ListCellRenderer to display weather info.

12) The main method will checks whether there is a command-line parameter (the hostname of the remote method hosting computer)

13) It then sets display properties through three method invocations.

14) It invokes the method setVisible to display the weather information.

15) Then you see the window which display the information from HTTP Server (obtained from Server)

(2) How does the client and server in a typical Java RMI application perform marshalling and unmarshalling operations? In the "WeatherService" RMI example which file(s) contain code that helps marshalling?

-------------- Server side:

1) When server construct the object using "WeatherServiceServer", it firstly gets data from HTTP Server.

2) The we initiated a buffer to temporarily store the raw data from HTTP Server

3) And then we scan the butter information line by line and extract the information what we need.

4) We extract exact data and assign to specific variable such as city, temperature, etc.

5) We store the prepared data into WeatherBean

6) We add each "Bean" to the List "weatherInformation"

7) We use UnicastRemoteObject.exportObject() to export the object → create a remote object reference that can be exported (to rmiregistry)

8) We let object "stub" to hold the exported object.

9) We rebind "stub" to the service name such as "WeatherService" → ready for client to lookup

10) We implement the interface method "getWeatherInformation()" for client.

11) Ok! So the "Bean Collection" weatherInformation is ready for client to fetch!!

12) You can say the above steps are so-called "marshalling"

→ These files help marshalling in the server side: "WeatherServiceServer.java", "WeatherService.java", "WeatherBean.java" (to store the data),

-------------- Client side:

1) The client invokes remote method getWeatherInformation (which was defined by server side earlier) of the interface WeatherService to obtain weather information, using information such as IP address and port number.

2) Get the registry object and search for remote object by using mutual agreed service name such as "WeatherService"

3) If we find the service name, we let "stub" to "point" to the remote object from server.

4) We get the data from server side by using “stub”’s method “getWeatherInformation”

5) We construct “weatherListModel” object and manipulate the data “weatherInformation” we get from server.

6) “weatherListModel” is a ListModel implementation. An object of this class contains WeatherBeans to be displayed in a JList.

7) The JList can retrieve elements only from a ListModel object. Thus, it adapts interface List to make it compatible with JList’s interface.

8) We uses a JList with a customerized ListCellRenderer to display weather info.

9) (b) The class uses a JList with a customerized ListCellRenderer to display weather info.

10) “WeatherCellRenderer” will use “WeatherItem”, which will combine the mapping table (varaibel to icon file) to find the icons.

11) We construct the windows with JFrame and set the window visible by “SetResizable” and “setVisible”