Deepak Nalla
Distributed Assignment #2
8/18/'16

PlaceServer.java

A "server" class, in this context, is the class which has a `main` method that creates an instance of the remote object implementation, exports the remote object, and then binds that instance to a name in a Java RMI *registry*.
This is done via the

- Naming.rebind("Places", new Places(null, null)); command which instantiates the Places(null,null) class

It also includes:

- Registry registry = LocateRegistry.getRegistry(port); which is used to start the RMI registry.

PlaceInterface.java

One of two interface definitions in this java program; the other being AirportInterface.java. This file extends the Remote object and allows us to implement Places file as an external remote object that we can call upon from the Client program. It consists of one additional method: the find_places function which accepts as parameters, two Strings - one for city/town name and the other for the State initials.

Places.java

This is the external remote object that is going to be cast and by implementing the PlaceInterface.java file.

- The function uses find_place(String name, String state) to receive the information as parameters which are going to be searched upon and compared to the binary file that is going to be parsed.
- It also uses another function createTable(Placelist placelist) which stores the information from the binary file source into a list <Place> datastructure
- It also imports the binary file information that was un-'tar'-ed using the protobuf compiler

Error cases

- If name is not found, return null
- If name is found in another state, then look at State
- If the state cannot return a match, return null

Client.java

We use this to connect to the two servers: PlaceServer and AirportServer. The client also binds to the rmiregistry to create the remote object and performs the lookup function for places to find lat, longitude of the places, and then it will enter that information into the lookup function for airports to return the 5 closest airports.

Error cases

- If no arguments provided or arguments are less than 2 when initially starting the client. As it needs the city name, and state to move forward in the process to find the latitude/longitude which will then be used to find the airports closest to it using find_airport(lat,lon).
- Additionally if no strings are found in the arguments, it will give another error asking you to re-enter the information.

==========

COMPILING AND RUNNING THE CODE

0. Make sure your CLASSPATH environment variable contains
your current working directory (.)

1. Compile the code
javac PlaceServer.java AirportServer.java Client.java AirportInterface.java PlaceInterface.java PlaceInfo.java AirportInfo.java Places.java Airports.java

2. You must run “rmiregistry (port number) & “ to run each server. Start the Place Server first, and then the Airports server (Make sure you use port number as 1099 as that is fixed value for Port for AirportServer, you can use any number for PlaceServer) or start in any order using the following command: (Make sure they are in separate Terminal windows)
Java PlaceServer 1299 (enter port number after file)
Java AirportServer 1099

4. run the client in yet another window:
java Client 23123 ABCDEF "this is just a test"
where the first argument is the port number of the rmi registry, second argument is name of place, and third argument is the state initials

5. Results of Airports shown in client window, as well as in AirportServer.