Group members: Gloria Leung and Snigdha Paka
Assignment 2

Instructions

Compilation

-Go to the folder that contains all the source files.
-Make sure that the CLASSPATH is set to the directory where all the files are located. Also make sure that the protobuf.jar file is in the CLASSPATH. This must be done for each of the four windows that you open.

For Example: export CLASSPATH=$CLASSPATH:.:protobuf.jar

-Compile the .proto files using: protoc *.proto --java_out=.
-Compile source files by running `javac *.java`

How to use

1. Start the registry in one window by typing:
   `rmiregistry <port> &`
2. Start the servers in two other windows by typing:
   `java PlaceServer <port>`
   `java AirportServer <port>`
3. Run the program in another window by typing:
   `java Client <hostname> <port> <city> <state>`

**All of the parameters are required.

Bugs/Peculiarities

Depending on the user input for the city, it is possible that the program will match to the wrong place. If the user enters a city, and there exists another city in the same state that starts the same, but is longer, the city may get matched to the longer name. For example, the user input of “Berkeley” in NJ may be matched to “Berkeley Heights township” in NJ. Most of these cases can be avoided by using a more specific version of the name, such as “Berkeley township” instead of just “Berkeley”.

Design Details

Before the client program starts running, the PlaceServer and AirportServer are started. They both read through the Google Protocol Buffer formatted files that contain the information about different locations as well as airports.

The PlaceServer initializes the Places class and registers it with the rmiregistry. The Place class reads though the places buffer and adds the details of each location into a PlaceInfo class which is then added to an Arraylist. Once that occurs, the Client is allowed to call the findplace method by sending in an input of a city name and the state abbreviation. A PlaceInfo object is returned to the client with data about the name of the city, the state where it’s located, its latitude and its longitude.

Each time the Airport server is started, the airports file is parsed and each airport entry is placed in a data structure, AirportInfo. All of the AirportInfo entries are placed in an ArrayList. Each time the Client has a query request, the latitude and longitude are given by the Places server, and the Client sends this information to the Airport server. The distance is calculated from each airport to the designated city (based on latitude and longitude). The airport and distance information is placed in AirportDistance, and all entries are in an ArrayList. Then, the built-in Java Collections sort is used to sort the the ArrayList entries by distance. The first five entries in the ArrayList are returned to the client.

Instead of allowing the hostname and port number to be optional, we require the user to enter the hostname and port name along with the city and state. The Client program will take in the user inputs and verify that they are correct. The program then checks that the services (remote objects) are available in the rmiregistry. It then proceeds to call the remote method findPlace() using the city and state that it received as arguments. After that, the Client calls the remote method findAirports() using the latitude and longitude of the place. Once it received the inputs, the Client prints out the five nearest airports.

Examples:
1) If the both Servers aren't running.
Client Window:
$java Client localhost 1099 Princeton NJ
There was a problem connecting to the servers

2) If only the AirportServer is running.
Client Window:
$ java Client localhost 1099 Princeton NJ
There was a problem connecting to the servers

3) If only the PlaceServer is running.
Client Window:
$ java Client localhost 1099 Princeton NJ
There was a problem connecting to the servers

4) If the places-proto.bin is missing.

Client Window:
$ java Client localhost 1099 Princeton NJ
Server Exception: places-proto.bin not found in Places

PlaceServer Window:
$ java PlaceServer 1099
binding //localhost:1099/Places

Missing places-proto.bin file: java.io.FileNotFoundException: places-proto.bin (No such file or directory)
New instance of Places created
server //localhost:1099/Places is running...

5) If the airports-proto.bin file is missing.

Client Window:
$ java Client localhost 1099 Princeton NJ
Server Exception: airports-proto.bin not found in Airports

AirportServer Window:
$ java AirportServer 1099
binding //localhost:1099/Airports
Missing airports-proto.bin file: java.io.FileNotFoundException: airports-proto.bin (No such file or directory)
New instance of Airports created
server //localhost:1099/Airports is running...

6) If the user mixes up the hostname and port name when running program.

Client Window:
$ java Client 1099 localhost Seattle WA
usage: java Client -h rmiregistryport -p port city state...
Please enter a number for the port

7) If user forgets an input parameter.

Client Window:
$ java Client localhost 1099 Seattle
usage: java Client -h rmiregistryport -p port city state...

8) If everything works normally.

Client Window:
$ java Client localhost 1099 "New York" ny
New York city, NY: 40.704234, -73.917927
code=LGA, name=New York/La Guardia, state=NY distance: 4.636775849606602 miles
code=JFK, name=New York/JFK, state=NY distance: 8.135552573927475 miles
code=TEB, name=Teterboro, state=NJ distance: 12.205921431374591 miles
code=EWR, name=Newark Liberty International, state=NJ distance: 13.198009701955616 miles
code=CDW, name=Fairfield, state=NJ distance: 22.11989272165418 miles

PlaceServer Window:
$ java PlaceServer 1099
binding //localhost:1099/Places
New instance of Places created
server //localhost:1099/Places is running...

AirportServer Window:
$ java AirportServer 1099
binding //localhost:1099/Airports
New instance of Airports created
server //localhost:1099/Airports is running…

9) If the rmiregistry has not been started

Client Window:
$ java Client localhost 1099 Seattle WA
There was a problem connecting to the servers

PlaceServer Window:
$ java PlaceServer 1099
binding //localhost:1099/Places
New instance of Places created
Place server failed:Connection refused to host: localhost; nested exception is:
java.net.ConnectException: Connection refused

AirportServer Window:
$ java AirportServer 1099
binding //localhost:1099/Airports
New instance of Airports created
Airport server failed:Connection refused to host: localhost; nested exception is:
java.net.ConnectException: Connection refused