Assignment 5 Summary

• Find the five airports closest to a given location
• One Client
• Two Servers
  – Place Server: get information about a location (latitude, longitude)
  – Airport Server: find airports near a given latitude, longitude
• Data is stored in Google Protocol Buffer format
  – Each server reads it at startup
Assignment

• The assignment uses Java RMI
• It does not have to be multithreaded
• You may work in groups up to 4
  – The larger the group, the more polished I expect your work to be
  – Group size > 1: submit a beautiful-looking project report

• You have a choice:
  – Write the assignment in Java & use RMI
    • You *must* use RMI if you write in Java
  – Write the assignment in Go
    • Use Google Protocol Buffers for communication between client and servers
These hints focus on a Java RMI implementation
Key Components

• The amount of code you will write is very small

• There are three parts that you need to get working
  1. Reading the places and airports databases
  2. Client-server communication
  3. Computing distances

• Any of these, especially 1 & 2, might cause confusion
• Start early
• Solve ONE problem at a time
• Then put it all together
Step 1

• Make sure you can read the Google Protocol Buffer files

• Download and compile the Protocol Buffers package
  – It’s the first link in the assignment

• You should generate:
  – Protocol Buffer compiler: protoc
  – A bunch of Java support classes
    • You can assemble them into one file: protobuf.jar
      
      cd protobuf-2.6.0/java/src/main/java/com/google/protobuf
      javac *
      jar cvf protobuf.jar com/

• Go through the tutorial – *ignore the assignment for now*
  – See the link: *Try the tutorial for your favorite language*
Step 1a: Tutorial

- The tutorial is in the *examples* directory in the source package
- The example is similar to what is needed for the assignment
  - Similar structures and examples of reading (and writing)
- If you cannot do the tutorial, you will not be able to do the assignment!

```
<table>
<thead>
<tr>
<th>Person</th>
<th>Place</th>
<th>Airport</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>state</td>
<td>state</td>
</tr>
<tr>
<td>id</td>
<td>name</td>
<td>name</td>
</tr>
<tr>
<td>email</td>
<td>lat</td>
<td>code</td>
</tr>
<tr>
<td></td>
<td>long</td>
<td>lat</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AddressBook</th>
<th>PlaceList</th>
<th>AirportList</th>
</tr>
</thead>
<tbody>
<tr>
<td>repeated Person</td>
<td>repeated Place</td>
<td>repeated Airport</td>
</tr>
</tbody>
</table>
```
Step 1b: Test program: Places

• Write a small program to read and print the list of places

```java
PlaceList pl = PlaceList.parseFrom(new FileInputStream(fname));
for (Place p: pl.getPlaceList()) {
    System.out.println(
        "state: " + p.getState() + " place: " + p.getName() + " lat: " + p.getLat() + " lon: " + p.getLon());
}
```

• Make sure protobuf.jar is in your CLASSPATH
• You should see output like

```
state: AL place: Abbeville city lat: 31.566367 lon: -85.2513
state: AL place: Adamsville city lat: 33.590411 lon: -86.949166
state: AL place: Addison town lat: 34.200042 lon: -87.177851
state: AL place: Akron town lat: 32.876425 lon: -87.740978
```
Step 1c: Test program: Airports

- Write a small program to read and print the list of airports

```java
AirportList al = AirportList.parseFrom(new FileInputStream(fname));
for (Airport a: al.getAirportList()) {
    System.out.println("state: " + p.getState() + " name: " + p.getName() + " code: " + p.getCode() + " lat: " + p.getLat() + " lon: " + p.getLon());
}
```

- Make sure protobuf.jar is in your CLASSPATH
- You should see output like

```
state: AL name: Anniston code: ANB lat: 33.58 lon: -85.85
state: AL name: Auburn code: AUO lat: 32.67 lon: -85.44
state: AL name: Birmingham code: BHM lat: 33.57 lon: -86.75
state: AL name: Centreville code: CKL lat: 32.9 lon: -87.25
```
Step 2a: Write a skeletal standalone program

• You know you can read the protocol buffer data

• Don’t worry about RMI for now

• Write standalone programs
  – Create *Places* and *Airports* classes (pick names you like)
  – Places
    • Constructor reads in the places database
    • main() can be a test function that takes a place name, looks it up, and prints results
  – Airports
    • Constructor reads in the airports database
    • main() can initially be a test function that looks up an airport
Step 2b: Refine the skeletal program

- Modify your airports test main() to look for closest airports
- Take latitude & longitude as parameters
- Find the 5 closest airports
  - Use the formula in the assignment to compute great circle distance
    \[ d = 60 \cos^{-1}(\sin(lat_1) \sin(lat_2) + \cos(lat_1) \cos(lat_2) \cos(lon_2-lon_1)) \]
  - You don’t need a clever algorithm
    - Just go through the list of airports
    - Compute the distance
    - See if each new distance should displace your list of \( n \) shortest distances
  - Print the results
    - Check that the results look right!
Step 3a: Make sure you can use RMI

• Again, ignore the assignment for now
• Download the RMI sample program (past recitation)
• Compile and run it
  – This will make sure you have no problems with RMI
  – … and no problems with CLASSPATH
Step 3b: Define Interfaces

• Define interface

• AirportsInterface (pick a name)
  – takes latitude & longitude and returns a list of airport info structures

• PlacesInterface (pick a name)
  – takes a place name and returns latitude & longitude
Step 3b: Create servers, client & add RMI

• Create servers for Airports & Places
  – Copy the sample RMI server
  – All it does is
    • Get a port from the command line
    • Instantiate the class
    • Register it with rmiregistry

• Your client will:
  – Call Naming.lookup to look up the Places & Airport servers
  – Places p = places.findplace(place_name)
  – AirportInfo closest[] airports.nearest(p.lat, p.long)
  – Iterate through the list and print the results
The End