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
    - Use Google Protocol Buffers for communication between client and servers
  - Write the assignment in Go
- Use Google Protocol Buffers for communication between client and servers

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
  - javac -c * -jar protobuf-2.6.0.jar
    - Also see: 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.
- If you cannot do the tutorial, you will not be able to do the assignment.

Step 1b: Test program: Places
- Write a small program to read and print the list of places.
- Make sure protobuf.jar is in your CLASSPATH.

```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());
}
```

Step 1c: Test program: Airports
- Write a small program to read and print the list of airports.
- Make sure protobuf.jar is in your CLASSPATH.

```java
AirportList al = AirportList.parseFrom(new FileInputStream(fname));
for (Airport a : al.getAirportList()) {
}
```

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.
      \[ d = 60 \cos^{-1} \left( \sin(lat_1) \sin(lat_2) + \cos(lat_1) \cos(lat_2) \cos(lon_1 - lon_2) \right) \]
    - 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