Chris Piech
CS 106A

Section #8
March 7, 2018

Solution to Section #8

Based on a problem by Brandon Burr and Patrick Young

1. Flight Planner Server

```
/*
 * File: FlightPlannerServer.java
 * ---------------------
 * A server program that, when run, reads in information
 * about available flights from a data file, and then listens
 * for incoming network requests.  This program can respond to
 * two types of requests:
 *
 * "getAllCities" -> we send back a list of all cities
 * "getDestinations" -> (needs parameter "city") we send back a
 *                      list of all cities reachable from the
 *                      provided city.
 */

import acm.program.*;
import acm.util.*;
import java.io.*;
import java.util.*;

public class FlightPlannerServer extends ConsoleProgram
    implements SimpleServerListener {

    /* The port number where we listen for requests */
    private static final int PORT = 8080;

    /* The name of the file containing our flight data */
    private static final String FLIGHT_DATA_FILE = "flights.txt";

    /* The server object that we use to listen for requests */
    private SimpleServer server;

    /* A map from city names to cities you can fly to from there */
    private HashMap<String, ArrayList<String>> flights;

    public void run() {
        readFlightData(FLIGHT_DATA_FILE);
        server = new SimpleServer(this, PORT);
        server.start();
        println("Starting server...");
    }

    /* Called when we receive a request to respond to */
    public String requestMade(Request request) {
        String cmd = request.getCommand();

        // Send back a list of all city names
        if (cmd.equals("getAllCities")) {
            println("Received getAllCities Request");
            ArrayList<String> cities = new ArrayList<String>();
            for (String cityName : flights.keySet()) {
                cities.add(cityName);
            }
            return cities.toString();

        // Send back a list of cities reachable from the provided city
        } else if (cmd.equals("getDestinations")) {
            String city = request.getParam("city");
            println("Received getDestinations Request for " + city);
            ArrayList<String> destinations = flights.get(city);

            /* If that city is not in our map, we need to make an empty
             * list because we cannot call toString on null.
             */
            if (destinations == null) {
                destinations = new ArrayList<String>();
            }
            return destinations.toString();
        } else {
            return "Error, cannot process request: " + request;
        }
    }

    /**
     * Reads in the city information from the given file and stores the
     * information in the HashMap of flights.
     */
    private void readFlightData(String filename) {
        flights = new HashMap<String, ArrayList<String>>();
        try {
            Scanner fileScanner = new Scanner(new File(filename));
            while (fileScanner.hasNextLine()) {
                String line = fileScanner.nextLine();
                if (line == null) {
                    break;
                }
                if (line.length() != 0) {
                    readFlightEntry(line);
                }
            }
            fileScanner.close();
        } catch (IOException ex) {
            throw new ErrorException(ex);
        }
    }

    /**
     * Reads a single flight entry from the line passed as an argument,
     * which should be in the form
     *
     * fromCity -> toCity
     *
     * Each new flight is recorded by adding a new destination city to
     * the ArrayList stored in our flights HashMap under the key for
     * the starting city.
     */
    private void readFlightEntry(String line) {
        int arrow = line.indexOf("->");
        if (arrow == -1) {
            throw new ErrorException("Illegal flight entry " + line);
        }

        // Note: trim() removes leading/ending spaces from a string
        String fromCity = line.substring(0, arrow).trim();
        String toCity = line.substring(arrow + 2).trim();
        defineCity(fromCity);
        defineCity(toCity);
        flights.get(fromCity).add(toCity);
    }


    /**
     * Defines a city if it has not already been defined. Defining
     * a city consists of entering an empty ArrayList in the flights
     * map to show that it has no destinations yet.
     */
    private void defineCity(String cityName) {
        if (!flights.containsKey(cityName)) {
            flights.put(cityName, new ArrayList<String>());
        }
    }
}
```

2. Flight Planner Client

```
/*
 * File: FlightPlannerClient.java
 * ------------------
 * A client program that talks to a flight server to allow
 * a user to plan out a flight path from a starting city
 * back to that starting city.
 */

import acm.program.*;
import java.io.*;
import java.util.*;

public class FlightPlannerClient extends ConsoleProgram {

    /* The network address for the flights server we should contact */
    private static final String HOST = "http://localhost:8080/";

    public void run() {
        println("Welcome to Flight Planner!");
        println("Here's a list of all the cities in our database:");
        ArrayList<String> cities = fetchCitiesList();
        if (cities == null) {
            println("Error: could not get list of all cities");
            return;
        }
        printCityList(cities);

        ArrayList<String> route = readInFlightRoute();
        if (route == null) {
            println("Error: could not get destinations");
            return;
        }
        printRoute(route);
    }
```

```
/**
 * Prompts the user for cities to travel to until they end in
 * the same city in which they started.  Returns null if we weren't
 * able to get a response for a network request.
 */
private ArrayList<String> readInFlightRoute() {
    println("Let's plan a round-trip route!");
    String startCity = readLine("Enter the starting city: ");
    ArrayList<String> route = new ArrayList<String>();
    route.add(startCity);
    String currentCity = startCity;

    while (true) {
        String nextCity = getNextCity(currentCity);
        if (nextCity == null) {
            // An error occurred
            return null;
        }
        route.add(nextCity);
        if (nextCity.equals(startCity)) {
            break;
        }
        currentCity = nextCity;
    }

    return route;
}

/**
 * Returns the list of all cities that the user can start at,
 * or null if we weren't able to get a response to our request.
 */
private ArrayList<String> fetchCitiesList() {
    try {
        // The getAllCities request needs no parameters
        Request request = new Request("getAllCities");
        String result = SimpleClient.makeRequest(HOST, request);
        return makeListFromString(result);
    } catch (IOException e) {
        return null;
    }
}

/**
 * Fetches all the cities the user could travel to from the given
 * city, and prompts them for a destination until they enter one
 * of these cities. Then returns the city they chose.  If we
 * weren't able to get a response for our request of destinations
 * for this city, this method returns null.
 */
private String getNextCity(String city) {
    ArrayList<String> destinations = fetchDestinations(city);
    if (destinations == null) {
        // An error occurred
        return null;
    }

    String nextCity = null;
```

```
        while (true) {
            println("From " + city + " you can fly directly to:");
            printCityList(destinations);
            String prompt = "Where do you want to go from "
                + city + "? ";
            nextCity = readLine(prompt);
            if (destinations.contains(nextCity)) break;
            println("You can't get to that city by a direct flight.");
        }
        return nextCity;
    }

    /**
     * Returns a list of cities that can be reached from the given
     * city. Returns null if we weren't able to get a response to our
     * request.
     */
    private ArrayList<String> fetchDestinations(String city) {
        try {
            /* The getDestinations request has a "city" parameter
             * that is the name of the city to get destinations for.
             */
            Request request = new Request("getDestinations");
            request.addParam("city", city);
            String result = SimpleClient.makeRequest(HOST, request);
            return makeListFromString(result);
        } catch (IOException e) {
            return null;
        }
    }

    /**
     * Prints a list of cities from the provided list. Each city name
     * is indented by a space.
     */
    private void printCityList(ArrayList<String> cityList) {
        for(int i = 0; i < cityList.size(); i++) {
            String city = cityList.get(i);
            println(" " + city);
        }
    }

    /**
     * Given a list of city names, prints out the flight
     * route, with a " -> " between each pair of cities
     */
    private void printRoute(ArrayList<String> route) {
        println("The route you've chosen is: ");
        for (int i = 0; i < route.size(); i++) {
            if (i > 0) print(" -> ");
            print(route.get(i));
        }
        println();
    }

    /**
     * (PROVIDED)
     * This is a wonderfully useful method that takes a list in string
     * form and turns it into and ArrayList. For example the string:
```

```
     *   "[cs106a, rocks, socks]"
     * will return an ArrayList with three elements:
     * "cs106a" "rocks" and "socks"
     */
    private ArrayList<String> makeListFromString(String listStr) {
        ArrayList<String> list = new ArrayList<String>();
        String raw = listStr.substring(1, listStr.length() - 1);
        String[] parts = raw.split(",");
        for(String part : parts) {
            String str = part.trim();
            if(!str.isEmpty()) {
                list.add(str);
            }
        }
        return list;
    }
}
```