Objective:
Build a multi-threaded server application.

What to do:
From the assignments page (http://josquin.cti.depaul.edu/~rburke/courses/w03/is313/hwk/), download the file hwk4.zip. This file contains mostly the same code as in homework #3 but with an additional package “client”. The client package contains three classes:

- TextModeClient: the main program, which is based on FFTextMode, except that it expects to communicate with the reservations database program over the network,
- RequestFactory: a helper class that creates FFRequest objects
- ClientRequest: a class that encapsulates the details of the interaction with the server.

TextModeClient works as is. However, there is no server for it to use. Your task in this assignment is to modify FFGUIMode so that it acts as a multi-threaded server application and can interact with TextModeClient. This will involve the following steps:

- Write a pair of classes ServerThread and WorkerThread. The ServerThread will accept connections on a java ServerSocket and create WorkerThreads to process them. The WorkerThread will interpret the requests, execute the relevant commands and transmit the results back to the client.
- It is strongly recommended that you encapsulate request processing in a ServerRequest class, analogous to the ClientRequest class in client package. This class would be responsible for reading request data from the client, extracting its properties and executing the necessary commands.
- Add a menu to the FFFrame user interface called “Server” with menu items “Start” and “Stop” that control the ServerThread. The Start and Stop menu items should be enabled and disabled according to the state of the server: that is, when no server thread is running “Stop” should be disabled; when the server is running, “Start” should be disabled.
- Since multiple threads may be modifying the database simultaneously, you must synchronize all the methods in the FrequentFlyerDB class that modify or examine the contents of the database.

When your program is working, you should be able to run FFGUIMode, turn on the server using the menu item and then run TextModeClient separately to interact with it. Your best bet is to run the client from the command line and HotelGUIMode from within your development environment, since you won’t have to modify the text mode code.

What to turn in:
Assemble your java files into a zip file named as in the following example: “RBurke_Hwk4.zip”. (Use your first initial and last name, of course.) Submit this file to the DLWeb website for the course.

Protocol:
The client program interacts with the server using a predefined protocol. The protocol consists of requests and responses, much like an HTTP interaction. A request is defined as a request name on a single line, followed by some number of request property in name/value pairs, like HTTP headers, followed by a single blank line, which indicates the end of the request. A response is simply a response type on a single line, followed by some number of lines contains the response body, following by a single blank line. The blank line is important, because this is how the client and server know that the end of the message has been reached. A request asking for the parameters of the capacity command would look like this:

(line 1) parameters
The command: name

The response would look like this:

OK
first
last

If a request is not understood by the server, it should return an error response type and then an error message in the response body. For example,

parameters
command: name

The response would look like this:

ERROR
Unknown command: name

The interaction with the client proceeds in the following protocol steps:

1. Client gets a command name from the user, sends a "prompts" request to server.
2. Server returns the prompts for the command.
3. Client sends a "parameters" request to server.
4. Server returns the parameter names for the command.
5. Now that the client has the parameter names and the prompts for each, it can prompt the user for the values needed to execute the command. When the values have been obtained, the client sends a request to the server for the execution of the command.
6. The server executes the command with the given parameters and returns the result.

Your server should handle the following request types:

- parameters (returns the parameters expected by the command)
- prompts (returns a set of prompts to use for the parameters)
- [capacity, add, all, delete, date, name, help (executes the appropriate command)]
- Note that you should not permit the remote client to force the server to exit. TextModeClient does not send “exit” to the server, it handles it locally, but a rogue client might.

**FFGUIMode application**

There have been several changes made to FFGUIMode to prepare it for your adaptation in this assignment. The most significant changes have been in the Command class:

- The Command class has been enhanced to support network interaction. Namely, a new mode constant has been defined “NET_MODE”. When a command is set to be in NET_MODE, it takes a Socket as an argument (similar to how establishing GUI_MODE required a JFrame argument). A new method displayResultsNet has been defined that allows output to the socket following the protocol described above.
- Commands that originate from the network cannot be created interactively (so there is no setParametersNet corresponding to setParametersGUI), so you will have to take the lines of the request, turn them into name/value pairs and set the parameters of the command individually.
The FFGUIMode application itself has been modified to take a port number as input. This is the port on which the server socket should listen in order to accept connections.

Hints and Notes:

- You may find it useful to create a "server" package in a subdirectory like the "client" package. This will segregate your server code from the rest of the application. If you do this, the only change you will have to make outside of the server package is to the FFFrame class, to create, start and stop your server.
- Your server class will need to make use of the CommandMap object in order to create new command instances. The most practical way to accomplish this is to pass the CommandMap to the constructor of the server class and store it as an instance variable.
- The approved way of stopping a thread that is waiting on an “accept()” call is to close the associated socket. This will throw a SocketException on the server thread, which your program must handle.
- You will find it useful to have your worker threads print status information to the console. The “echo mode” in the TextModeClient lets you turn this output on and off in the client application. You may want to implement a similar mechanism in the server.
- Both the TextModeClient and FFGUIMode clients now accept a argument that is the port number on which communication will take place. This port number must be the same for the programs to talk to each other. Example:
  ```
  > java –cp "." ui.HotelGUIMode 2233
  In a different command shell,
  > java –cp "." client.TextModeClient 2233
  ```
  It is assumed in the TextModeClient that the server will be running on the same computer. If you want to test it on different computers, you will have to change the argument to the RequestFactory constructor from “localhost” to the name of the actual host computer. Beware of firewalls, however!
- It is VERY IMPORTANT that you use “flush” to transmit data from your server after you have written it to the socket’s output stream, or you can use the autoflush option in the constructor for PrintWriter. Otherwise your data will sit in the server’s I/O buffer and the client will never see it.
- You will find it best to shut down your server application in stages, first stopping the server thread and then killing the application. If you do not, it will take awhile for the operating system to realize that your server is no longer listening and this may mean that you have to restart the server using a different port number. (If you’re feeling ambitious, you could integrate shutting down the server thread into the windowClosed event handler for HotelFrame.)
- The run method for the server thread will have the following pseudo-code (omitting exception handling):
  ```
  create a server socket
  while active
      attempt to accept a new connection
      create a new worker thread for new connection
      start the thread
  ```
- The run method for a worker thread will have the following pseudo-code (again without exception handling):
  ```
  create a request object from the input stream of the socket
  execute the request object (which may involve instantiating a Command object and executing it)
  ```
- If you are having difficulty debugging your server code, you may find it useful to interact with it via telnet. You can enter the lines of the protocol manually, not omitting the blank line at the end, and see what the server returns.