Post-It Board Protocol
CP372 Assignment 1
Request for Comments
Group No. 053

Binoli Joshi - 160456810
Charmi Desai - 160491350

Post-It Board Protocol - PIBP

Status of this Memo

This memo specifies a protocol for the Post-It Board application and requests discussions and suggestions for improvement. The Post-It Board application is a client-server application that simulates an office message board and will be implemented using the latest version of Java and the Java's stream socket API.

Table of Contents

*Changes to RFC are in red

1. Introduction

1.1 Purpose

The Post-it Board Protocol is an application-level protocol for a client-server application that simulates an office message board. The server's primary purpose will be to store post-it notes and the client's purpose is to create as well as request notes. The client can connect to the server through a GUI and multiple clients can connect to the single server. This project will help in understanding how clients and servers interact with each other.

1.2 Terminology

Client - An application program that accesses a service made available by a server and sends requests to the server.

Server - An application program that accepts requests from the client and sends back a response. It provides service to one or many clients.

Post-It Note - The note that client wants to post to the server. The post-it note is instantiated through a POST message request from a client

Message - Requests that the client can make to the server. This protocol allows for five types of message requests to the server: POST, GET, PIN/UNPIN, CLEAR, DISCONNECT

Board - The office message board that visually shows the post-it notes mapped on the message board. Each post-it note, upon initialization contains x and y coordinates as well as note width and height which specify where the note is on the message board.

1.3 Requirements

The client must request a connection with the server and must be able to send five types of messages to the server. The five types include:

1. POST <note> - requests the server to add a new note.
    - <note> - represents the notes containing the messages like a post-it, it includes all the details of the note including location coordinates (based on the lower-left corner), width, height, colour of the note, the status (pinned/unpinned) and the message it contains.
2. GET <request> - requests the server to send back all notes that satisfy the properties of the request.
    - <request> - represents the details of the notes that the client requires. It can equal:
      - PIN - coordinates of all pins are returned
      - color=<color> - all notes that are of <color> color are returned
      - contains=<data> - all notes that contain the point <data> are returned

- refersTo=<string> - all notes that contain the substring <string> in the message are returned

3. PIN/UNPIN <data> - requests the server to add or remove the notes corresponding to data, to message board.
    - <data> - is a pair of integers that represent the x and y coordinate of a point on the board.
4. CLEAR - request the server to forget all notes which are not pinned.
5. DISCONNECT - requests the server to disconnect from this client.

The server must accept requests for connection from multiple clients and hold the data that it receives from the clients. It must also respond to all the types of messages listed above.

1.4. Overall Operation

The Post-It Board Protocol is based on a request/ response system. It will allow users to alter a data structure containing information about note messages in the server. The user will do this by running a client and connecting to the server. The user will use a GUI to interact with the server. Using the GUI, the user can post notes to the board, get specific notes stored by the server, pin and unpin notes to the board. Additionally, the user can clear all notes not pinned and disconnect the client from the server. The client will perform error checking to ensure that the request sent by the user follows the required format. Then the server will receive the request in the form of a string and will go through the process of acquiring the data required to respond to the request. The server will respond to client's request through the GUI.

2. Responsibilities

2.1 Server Responsibilities

Server responsibilities include accepting a connection from the clients and being able to connect to multiple clients at the same time. The server has to read the message request from the clients and alter the data structure, containing the pinned notes, accordingly. Upon server startup, the data structure is empty. This means that restarting the server will erase everything in the data structure. If an error occurs, the server must communicate that to the client and the user. If the request is completed, the appropriate response should be sent to the user.

2.2 Client Responsibilities

Client responsibilities include requesting a connection to the server, sending user input (requests) to the server and providing a GUI for user interface. The GUI should include all possible requests and the appropriate fields for input. The GUI should also include a section for error messages and responses from the server to be displayed. The GUI also shows a visual representation of the pinned and unpinned notes on the server. It should be noted that it is the client's responsibility to store unpinned notes (as the server only stores notes that are pinned). Hence, when a client disconnects from the server, the client's unpinned notes disappear along with it.

3. Format Of Messages

3.1 Format Of Notes

The Notes object represents a post-it note and is represented in the program as: Notes = (int xCoordinate, int yCoordinate, int width, int height, string colour, string message, int status). The status attribute indicates whether or how many times a note has been pinned. Status = 0 is the default and means the note is unpinned. These parameters indicate how the post-it is to be displayed on the message board.

3.2 Client Request

The client requests will be sent as a string to the server where each element will be separated by a comma. For example, request formats are as follows:

POST,2,3,10,20,white,Meeting next Wednesday from 2 to 3
GET,PINS
GET,color=color
GET,contains=x&y
GET,refersTo=string
GET,color=somecolor,contains=x&y
GET,color=somecolor,refersTo=something
GET,contains=x&y,refersTo=something
GET,color=somecolor,contains=x&y,refersTo=something
PIN,x,y
UNPIN,x,y

The CLEAR and DISCONNECT requests are buttons on the GUI.

3.3 Server Responses

For POST, PIN, UNPIN, CLEAR and DISCONNECT requests, the server will respond with a message stating if the action was successful or not. The server will return only the message part of the appropriate notes for a particular GET request separated by commas.
For example, GET color=white returns:
Meeting next Wednesday from 2 to 3, Pick up Fred up from home at 5,

4. Synchronization Policies

Due to the fact that the application may have several clients accessing the server at a time, there is a need for synchronization policies to ensure that changes that one client makes on their GUI, are reflected on other clients' GUI. Additionally, synchronization policies protect the program against unforeseen results when two different clients try accessing the same resource on the GUI at the same time. For example, suppose that there exists an unpinned note on the server and client A tries to pin that note on the server while at the same time, client B tries to delete the note from the server.

The resources that need to be synchronized in this application include the data structures that will be used to store the pinned and unpinned post-it notes (Refer to section 6 for data structure information). Fortunately, Java provides thread synchronization which ensures that only one thread can access a resource at a given point in time. To allow several clients access to the contents of the message board, the program will use threads to concurrently execute the five types of messages to the server. Each of these threads will be what Java calls, synchronized – this ensures that only one thread has access to a resource at a given time. Java does this by using locks – ex. when one thread wants access to a given resource, it obtains a lock on the resource before it executes the method in the thread and if a thread tries to access a resource that another thread is using, the thread is blocked until there is no longer a lock on the resource. This way we synchronize the actions of multiple threads and ensure that only one thread is accessing the pinned and unpinned data structures at a given point in time. Therefore, it can be assured that the application is thread safe and that multiple clients can access the server and see the same post-it notes on the board.

5. Border-Case Behavior

This section deals with edge cases and how the program will respond to them.

Empty Board
In the event that a client sends a GET request to the server which contains an empty board (meaning there are no pinned notes to retrieve) the server will send the client a message that says, “No pinned notes to retrieve - board empty!)”. On the client side, this will be reflected as an error message in the GUI.

Server Unexpected Shutdown/ Crash
In the event that the server crashes, the server will lose all the data structures attached to it and hence all the pinned notes will be lost.

POST Request Issues
If a client tries to send a POST request in which the message section is too long of a string for Java to store, the server will send the client an error message that says “Message string too long! - Note was not created!” and discard the note. If a client tries to send a POST request that contains empty or incomplete parameters, the server will send the client an error message that says “POST request format is incorrect - Note was not created!” and discard the note.

Requests for Non-existing Pins

In the event that a client tries to UNPIN a pin a note that does not exist on the server, the server will send the client a message that says “Nothing to UNPIN - PIN does not exist”. Similarly, if a client sends the server a GET request which matches none of the pinned notes (meaning there are no notes that match the client’s specifications), the server will send the client a message that says “No notes to retrieve”.

6. Data Structure Used

The data structure will hold all the notes and their details, and essentially will act as a dictionary. The ArrayList data structure in Java will be used to store both pinned and unpinned notes. The pinned notes will be stored as an ArrayList on the server as they need to be accessed by all clients. The unpinned notes will also be stored as an ArrayList but on the client side of the program as unpinned notes do not need to be accessed by other clients. This is the most appropriate of the possible structures since it allows the objects to have attributes. The server will search through the ArrayList to find the information needed to respond to client requests.

7. Errors

7.1 Client Errors

The following is a list of errors the client will handle:

- Incorrect IP address or port number entered
- x-coordinate and y-coordinate are outside the parameters of the message board
- Missing fields in the POST request
- Incorrect user input for any request

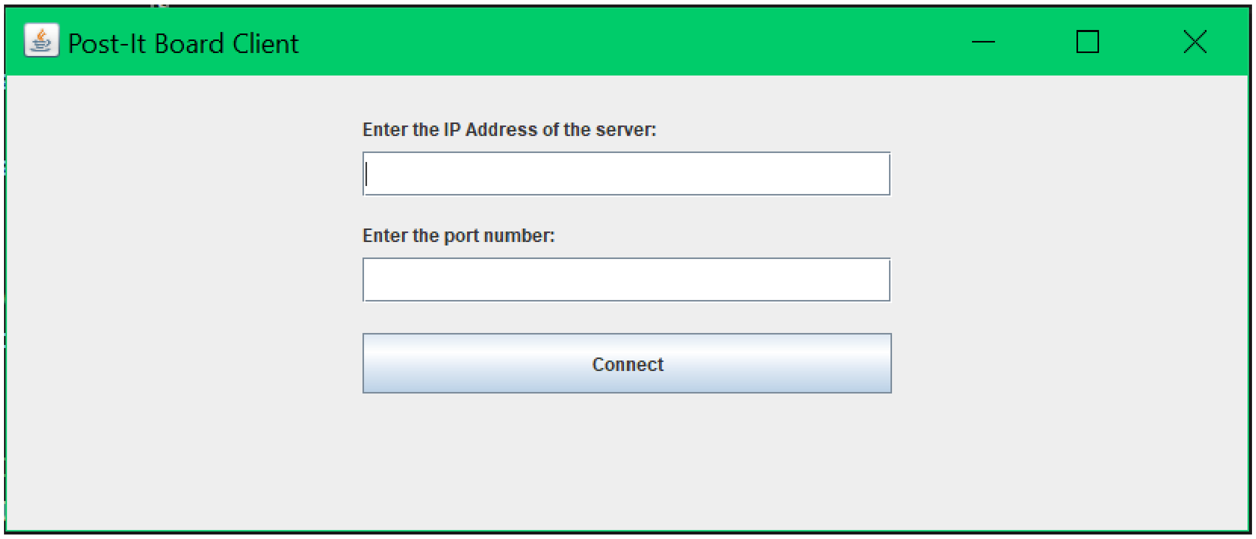
7.2 Server Errors

The following are a list of errors that the servers will handle:

- Connection failed
- Request for a note that does not exist

8. GUI

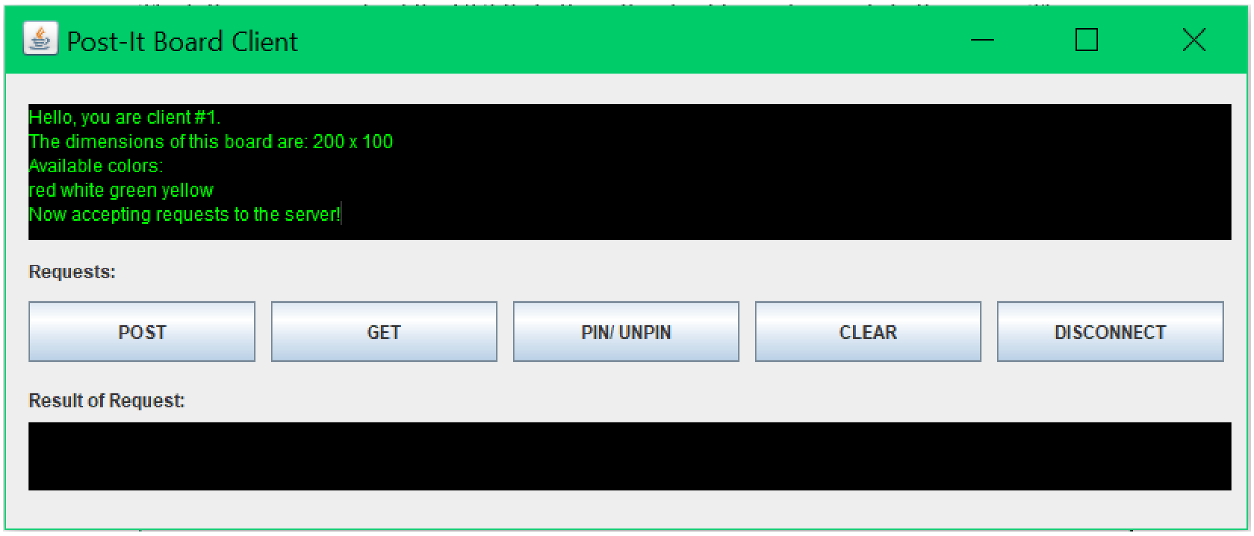
The following is the GUI that allows the client to send requests to the server.



Post-It Board Client

Enter the IP Address of the server:

Enter the port number:



Post-It Board Client

```
Hello, you are client #1.
The dimensions of this board are: 200 x 100
Available colors:
red white green yellow
Now accepting requests to the server!
```

Requests:

Result of Request: