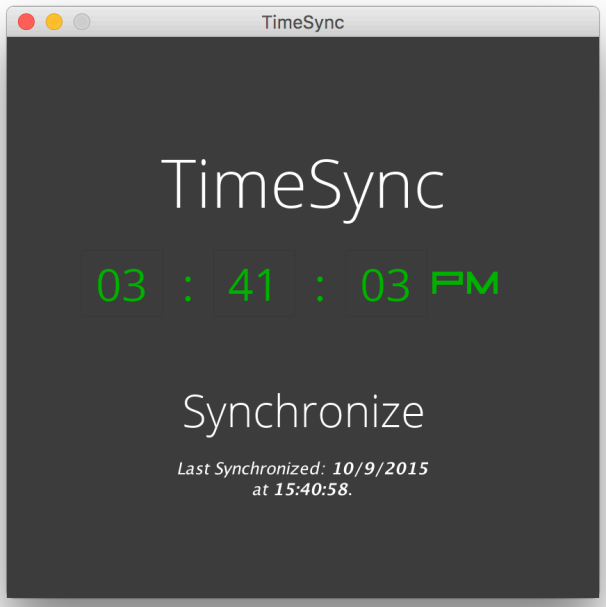
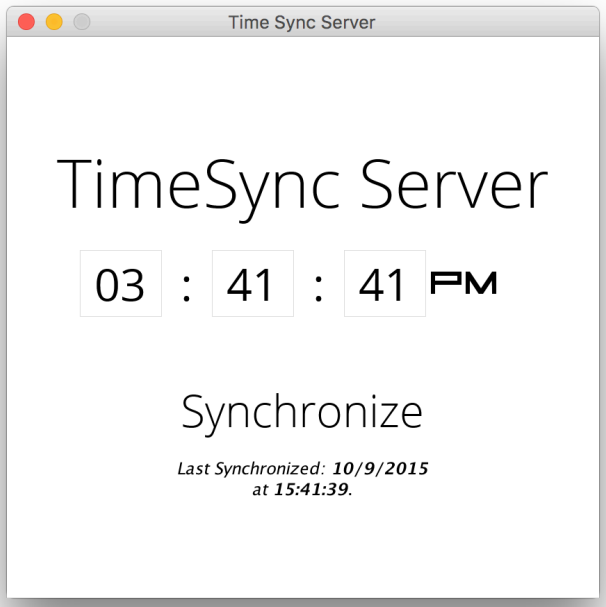
TimeSync

Sam Dindyal | Balin Banh | Daniel Tran





TimeSync is a simple Java application, written with the `javax.swing` package and as an assignment for Computer Networks I *(CPS706)*. It is composed of two parts, a client side and a server side; both of which feature their own user interface. The client and server communicate through the TCP protocol can selectively sync time and/or date, which is determined on the server side application.

In order to run TimeSync, run `TimeSyncServer` first, and, with that running, run `TimeSyncClient`.

Please note that TimeSync ***requires JRE 8 or newer.***

OVERVIEW

In this implementation of a Client/Server application, `TimeSyncServerRuntime.java`, `TimeSyncRuntime.java`, and `DateTimePanel.java` are the main components of the system. The objective of DateTimePanel.java is to keep track of time, and present it in a GUI.

`TimeSyncRuntime.java` and `TimeSyncServerRuntime.java` is the client/server representation within the application. Using TCP, `TimeSyncServerRuntime.java` waits opens a socket and waits for a response from the client.

`TimeSyncServerRuntime.java`

```
private ServerSocket serverSocket;
```

```
serverSocket = new ServerSocket(TimeSyncLibrary.TCP_SERVER_SOCKET);
```

These two lines of code create a new TCP socket.

The `TimeSyncRuntime.java` then sends a response with a buffered reader and closes the socket. After which, `TimeSyncServerRuntime.java` opens a new socket, and sends the requested data back to `TimeSyncRuntime.java` and closes the socket.

The client and server both run on two separate threads, to avoid interfering with the UI.

`TimeSyncRuntime.java`

```
private void connect() {
    try {
            clientSocket = new Socket(location, TimeSyncLibrary.TCP_SERVER_SOCKET);
            inputFromServer = new BufferedReader(new InputStreamReader(clientSocket.g
etInputStream()));
            outputToServer = new DataOutputStream(clientSocket.getOutputStream());
        } catch (Exception e){e.printStackTrace();}
    }
```

The code snippet above creates a new TCP socket to receive input and to output to the server.

```
try {
        syncThread.join();
        System.out.println("SERVER RESPONSE: " + serverInput);
        listener.actionPerformed(new ActionEvent(this, ActionEvent.ACTION_PERFORMED,
"", System.currentTimeMillis(), 0));
    }catch(Exception e){e.printStackTrace();}
```

This code snippet is responsible for completing the syncing process between the client and the server. Because we are using multiple threads, we need to ensure that everything stays synchronized.

`TimeSyncServerRuntime.java`

```
public void start()
{
    System.out.println("Server started.");
    try {
            listen();
            connect();
            respond(clientInput);
            connectionSocket.close();
        }catch(Exception e) {e.printStackTrace();}
}
```

In the code snippet above, the server is waiting for a request from the client. It closes the socket after responding.

```
private void connect ()
{
    try {
            connectionSocket = serverSocket.accept();
            outputStream = new DataOutputStream(connectionSocket.getOutputStream());
            inputReader = new BufferedReader(new InputStreamReader(connectionSocket.g
etInputStream()));
        } catch(Exception e){e.printStackTrace();}
}
```

The code in the snippet above is responsible for opening a socket for the client to send a request.