Práctica 5

Sockets Cliente

Programación Orientada a Objetos

Humberto Alejandro Ortega Alcocer

Opción: 5 “ChatBot Básico o NanoAlexa”

14 de diciembre de 2020

Grupo: 2CM1

Introducción

En cuanto nos interesamos por aprender desarrollo orientado a objetos, nos daremos cuenta de que una de las grandes ventajas es que podemos modelar sistemas distribuidos de forma rápida y eficiente. Para la práctica actual, estamos trabajando con un programa muy simple: un chatbot. Un "*chatbot*" se define como un programa de computadora capaz de responder preguntas y mantener una conversación con el usuario. Cabe recalcar que si bien existen productos extraordinariamente avanzados que, mediante Inteligencia Artificial realizan la aproximación de la mejor respuesta posible, en nuestro caso la intención parte más de conocer el comportamiento, diseño y detalles técnicos al implementar un chatbot "simple", es decir, uno dónde tengamos una serie de preguntas y respuestas predefinidas.

Es importante entender el objetivo de nuestra práctica, el cual es que podamos visualizar de forma clara el proceso de desarrollo y control que conlleva hacer software distribuido en red. Siendo nuestra prioridad inicial poder establecer dicha conexión entre dos o más clientes, nos enfocaremos en poder ofrecer una experiencia de chat sólida dónde podemos aumentar las capacidades de nuestro intento de Inteligencia Artificial conforme se vea apropiado.

El concepto detrás de realizar un chatbot, siempre parte de la curiosidad y le ingenuidad humana. Ese sentimiento y razón para sentirnos conectados es la razón que nos invitan a pensar en programas como este, dónde lo que buscamos es que la gente pueda platicar e interactuar con nuestro software. Para comenzar la interacción, como en cualquier sala de chat de IRC (allá por los 2000's), el usuario seleccionará su *"nickname"*, el cual se utilizará para mostrar el origen de los mensajes. Posteriormente, el usuario hace click en un botón para conectarse con el servidor y, en caso de que se haya establecido una conexión exitosa, se inicia el

proceso infinito de chat estándar en dónde el usuario puede escribir y enviar mensajes, así como recibirlos y leerlos.

Nuestra parte de "*Inteligencia Artificial*", así, entre comillas, es que, en caso de recibir un mensaje externo con una palabra clave, se realizará una respuesta inmediatamente en función de una tabla de referencia dónde podremos, manualmente, colocar algunas respuestas rápidas, simulando un agente de Inteligencia Artificial a su más mínima expresión.

El planteamiento de la práctica, en palabras del profesor Roberto Tecla, es:

3.- **ChatBot Basico o Nano Alexa** (Carpeta estatiProgBarRedSimBot)
Importante si hace esta opcion no tiene que modificar el servidor (VerySimpleChatServer) ni escribir uno nuevo.

Modificar el programa que cambia el estado de animo del tamagochi en la maquina local y la maquina remota de modo que: un cliente y otro cliente interactúen del siguiente modo un cliente envía una pregunta y otro cliente le envía la respuesta. Cada cliente puede almacenar al menos 10 preguntas y 10 respuestas predefinidas (se pueden usar 2 arreglos o un HashMap). Si se le pide a otro cliente que cuente un chiste entre 10 chistes disponibles elije uno al azar y lo envia Si se le pide a otro cliente la hora entonces que envie la hora actual.

En el problema 1 el objeto que se mueve entre maquinas es de la clase Icono dicha clase contiene un entero que es el dato que nos interesa enviar de una maquina a otra.

```
public class Icono implements Serializable {
    int turno;
    public Icono(int turno){ this.turno = turno; }
    public int getTurno(){ return turno; }
}
```

En este caso se debe enviar un objeto que contenga una cadena (mensaje) al servidor y se debe enviar si dicho mensaje es una peticion o una respuesta es decir se incluir en el objeto enviado el tipo del mensaje. Si llega una peticion se envia una respuesta. Si llega una respuesta se muestra.

Ejemplos de preguntas tipo y respuestas tipo

En que **ciudad** vives? **D.F.**
Cuántos **años** tienes? **20**
En que **escuela** estudias? **ESCOM**
Dame la **hora**: Son las (poner hora actual aqui)
Cuenta un **chiste**: **Habia un perro de goma que cuando se rascaba se borraba.**
Cuenta un **chiste**: **¿Los hacendados hacen dados?**

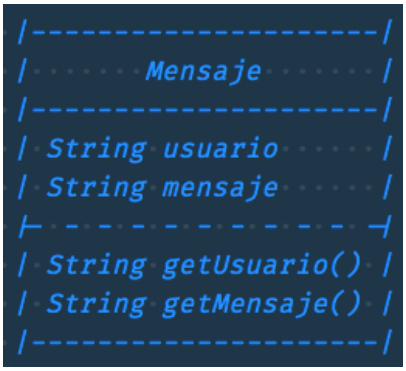
Desarrollo

Para comenzar, lo primero que tengo que describir es que mi programa **no utiliza la interfaz del tamagotchi**, esto porque en mi entorno de desarrollo me es imposible instalar Java3D por lo que el presente chat emplea, de los archivos previamente proporcionados por el profesor para otras prácticas:

- VerySimpleChatServer.java
- Red.java
- LeeRed.java
- IncomingReader.java

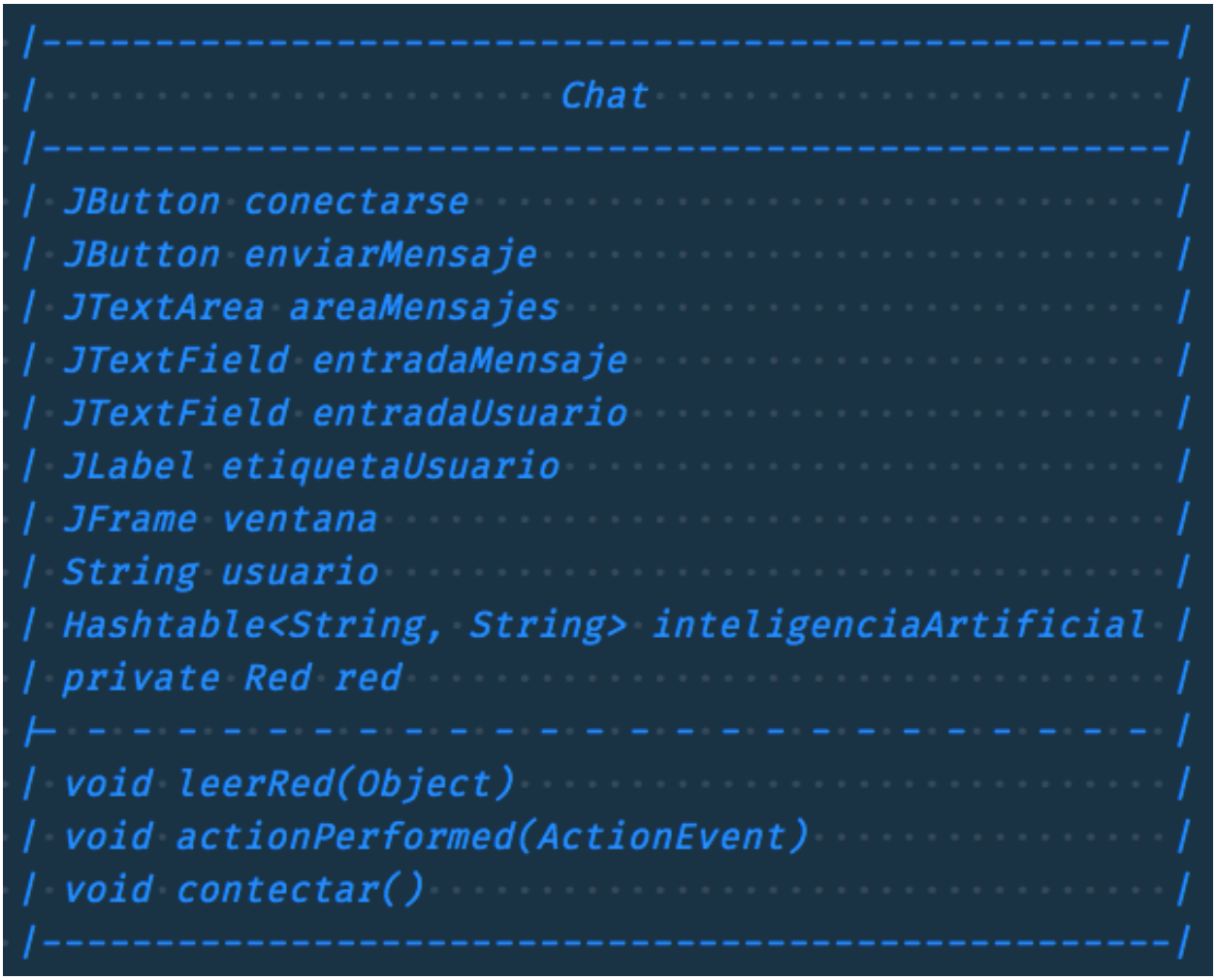
Con estos archivos, es más que suficiente para montar una aplicación de chat y no ahondaré en detalles de su implementación sino que me concentraré en mostrar el contenido de mi práctica.

Ahora sí, hablando de la parte particular a mi práctica, lo primero que tenemos que visualizar es el **Mensaje** que vamos a recibir y a enviar. Para esto, se propone el siguiente diagrama UML:



El cual realmente es muy simple, tendremos una variable *usuario* en la cual almacenaremos el nombre del usuario que envió dicho mensaje y una variable *mensaje* en la cual almacenaremos el cuerpo del mensaje en cuestión. Si bien, la

clase Mensaje carece de gran ciencia, es esta clase la que representa nuestra principal interacción con el usuario, es decir, los mensajes del chat.
Lo siguiente que analizaremos es el Chat en sí mismo. Si bien, el chat no presenta mucha funcionalidad, es importante que analicemos el diagrama UML propuesto para el mismo:



Comenzaremos describiendo las variables miembro que corresponden a elementos gráficos. A continuación se hace una visualización de cada uno de los elementos así como su interés y razón de existir dentro de nuestra ventana:

- *etiquetaUsuario*: Se refiere a una etiqueta con el mensaje “Nombre de Usuario: “ para indicarle al usuario el motivo del campo siguiente.

- *entradaUsuario*: Se refiere al campo de texto para ingresar el nombre del usuario en la aplicación.
- *Conectarse*: Se refiere al botón para realizar el proceso de conexión con el servidor.
- *areaMensajes*: Se refiere al área dónde se irán “imprimiendo” o mostrando, los mensajes que sean recibidos y emitidos por nuestro programa.
- *entradaMensaje*: Se refiere al campo de entrada para que el usuario pueda redactar el mensaje a enviar.
- *enviarMensjae*: Se refiere al botón que utiliza el usuario cuando éste desea enviar un texto.

Como se puede observar, en realidad estos elementos no son muy relevantes y su interacción más grande se encuentra dentro del constructor de la clase Chat cuya implementación se muestra en la siguiente página:

```java
// Constructor de la clase Chat.
public Chat() {
  // Creamos la ventana.
  this.ventana = new JFrame("Chat - Práctica 5 - POO ESCOM");
  this.ventana.getContentPane().setLayout(new BorderLayout());

  // Creamos los páneles para maquetar la interfaz.
  JPanel p1 = new JPanel(), p2 = new JPanel(), p3 = new JPanel();
  p1.setLayout(new BorderLayout());
  p2.setLayout(new BorderLayout());
  p3.setLayout(new BorderLayout());

  // Creamos los componentes de la interfaz.

  this.areaMensajes = new JTextArea("Presiona \"Conectarse\" para iniciar...", 10, 40);
  this.areaMensajes.setEditable(false);

  // Creamos un JScrollPane para que se vean los mensajes de forma infinita.
  JScrollPane scroll = new JScrollPane(this.areaMensajes);
  scroll.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);

  this.conectarse = new JButton("Conectarse");
  this.conectarse.addActionListener(this);

  this.entradaMensaje = new JTextField(20);
  this.entradaMensaje.setEnabled(false);

  this.entradaUsuario = new JTextField("Octavio Paz");

  this.etiquetaUsuario = new JLabel("Usuario: ");

  this.enviarMensaje = new JButton("Enviar mensaje");
  this.enviarMensaje.addActionListener(this);
  this.enviarMensaje.setEnabled(false);

  // Agrupamos nuestra interfaz.
  p1.add(this.etiquetaUsuario, BorderLayout.WEST);
  p1.add(this.entradaUsuario, BorderLayout.CENTER);
  p1.add(this.conectarse, BorderLayout.EAST);

  p2.add(scroll, BorderLayout.CENTER);

  p3.add(this.entradaMensaje, BorderLayout.WEST);
  p3.add(this.enviarMensaje, BorderLayout.EAST);

  // Añadimos nuestros páneles a la ventana.
  this.ventana.add(p1, BorderLayout.NORTH);
  this.ventana.add(p2, BorderLayout.CENTER);
  this.ventana.add(p3, BorderLayout.SOUTH);

  // Ajustamos operación de ventana.
  this.ventana.pack();
  this.ventana.setVisible(true);
  this.ventana.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

  // Inicializamos nuestra "inteligencia artifical".
  this.inteligenciaArtificial = new Hashtable<String, String>();

  // Añadimos... inteligencia artificial.
  this.inteligenciaArtificial.put("¿en qué ciudad vives?", "Ciudad de México");
  this.inteligenciaArtificial.put("¿cuántos años tienes?", "23");
  this.inteligenciaArtificial.put("¿en qué escuela estudias?", "ESCOM");
  this.inteligenciaArtificial.put("¿cómo se llama tu perro?", "Maddie");
  this.inteligenciaArtificial.put("¿qué materia no quieres reprobar?", "POO, de entrada");
  this.inteligenciaArtificial.put("¿cuál es tu película favorita?", "El Padrino I");
  this.inteligenciaArtificial.put("¿qué música te gusta?", "Reggaetón no");
  this.inteligenciaArtificial.put("¿cómo se llama tu hermano?", "Gerardo");
  this.inteligenciaArtificial.put("cuenta un chiste",
      "Había una vez un perro de goma que cuando se rascaba se borraba :(");
  this.inteligenciaArtificial.put("otro chiste", "¿Los hacendados hacen dados?");
}
```

Como se puede apreciar en la imagen, no hay nada muy relevante en cuanto al aspecto gráfico de la aplicación se refiere, sin embargo existe un tema a discutir, y es el de la “Inteligencia Artificial”. Para implementar esta, utilicé un HashTable, esto porque es un método de almacenamiento simple y rápido que nos permite guardar valores *key-value* en dónde, al tener un “key” (llave) válido, se accederá de forma casi instantánea a su valor. Esto es un caso de uso perfecto para cuando queremos recibir un mensaje, verificar si existe dentro de nuestras llaves y, en caso de que exista, responder con la respuesta correspondiente a la llave en cuestión.
Si bien este procedimiento no es avanzado ni mucho menos, nos permite de forma rápida y eficiente implementar esa interacción entre el usuario y cierto nivel de interacción artificial de forma eficiente.
Para enviar un mensaje y para determinar si ya nos hemos conectado previamente al servidor, utilizamos el método *“actionPerformed”* con el cual determinaremos si el botón que fue utilizado fue el botón “Conectar”, lo cual nos indicaría que el programa acaba de iniciar y procederíamos a inicializar la conexión para posteriormente configurar la interfaz gráfica de usuario de la forma apropiada para que nuestro usuario pueda comenzar a enviar mensajes. En caso de que el botón de origen sea el botón de “enviar mensaje”, esto nos indicaría que ya tenemos una conexión establecida y que el usuario ya se encuentra en un punto para empezar a utilizar nuestro programa. La implementación de la función es:

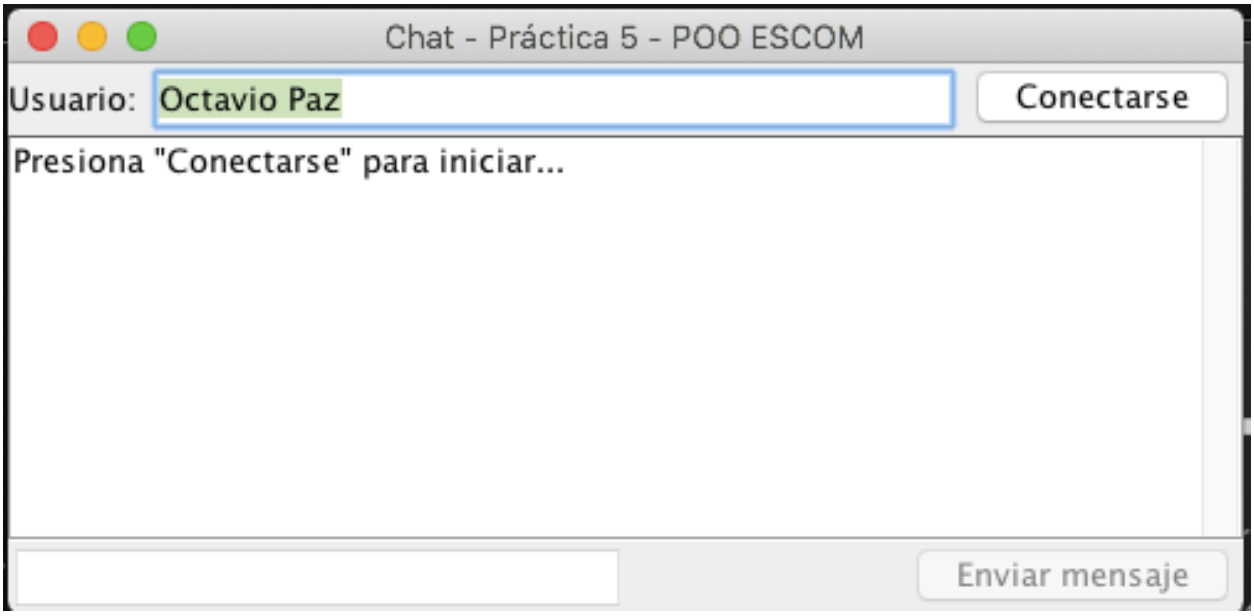
```java
// Método que es llamado cada que se haga click en un botón.
public void actionPerformed(ActionEvent evento) {
  // Extraemos el botón de orígen.
  JButton botonSeleccionado = (JButton) evento.getSource();

  // Determinamos el flujo a seguir.
  if (botonSeleccionado == conectarse) {
    // Si es el botón conectar, primero nos conectamos al servidor.
    this.conectar();

    // Mostramos confirmación en nuestra área de mensajes.
    this.areaMensajes.append("\n" + "¡Conectado correctamente! Ya puedes chatear.");

    // Obtenemos el nombre de usuario seleccionado.
    this.usuario = this.entradaUsuario.getText();

    // Desactivamos la entrada del usuario.
    this.entradaUsuario.setEnabled(false);

    // Desactivamos el botón de conectarse.
    this.conectarse.setEnabled(false);

    // Activamos la entrada para escribir mensaje y el botón enviar.
    this.entradaMensaje.setEnabled(true);
    this.enviarMensaje.setEnabled(true);
  } else {
    // Si es el botón enviar, mandamos el mensaje en red.
    red.escribeRed(new Mensaje(this.usuario, this.entradaMensaje.getText()));

    // Añadimos el mensaje a nuestro cuadro de texto.
    this.areaMensajes.append("\n" + this.usuario + " (Yo): " + this.entradaMensaje.getText());

    // Limpiamos el campo.
    this.entradaMensaje.setText("");
  }
}
```

Asi también , podemos observar que, en caso de que nosotros seamos los que enviamos el mensaje en esta función se añade a nuestra ventana central de chat dónde podemos visualizar todos los mensajes que han sido enviados.
El siguiente punto a considerar es cuando recibimos un mensaje, esto porque existen dos distintos caminos para reaccionar a un mensaje entrante. El primer camino es cuando el mensaje se refiere a algo que nuestra semi-Inteligencia Artificial no puede “reconocer”, es decir cuando el mensaje que hemos recibido no es igual a alguna de las llaves ingresadas en el HashTable, en dónde lo que se hace es imprimir el mensaje y ya. El segundo camino es dónde encontramos que el mensaje entrante corresponde a un elemento presente en alguna de nuestras

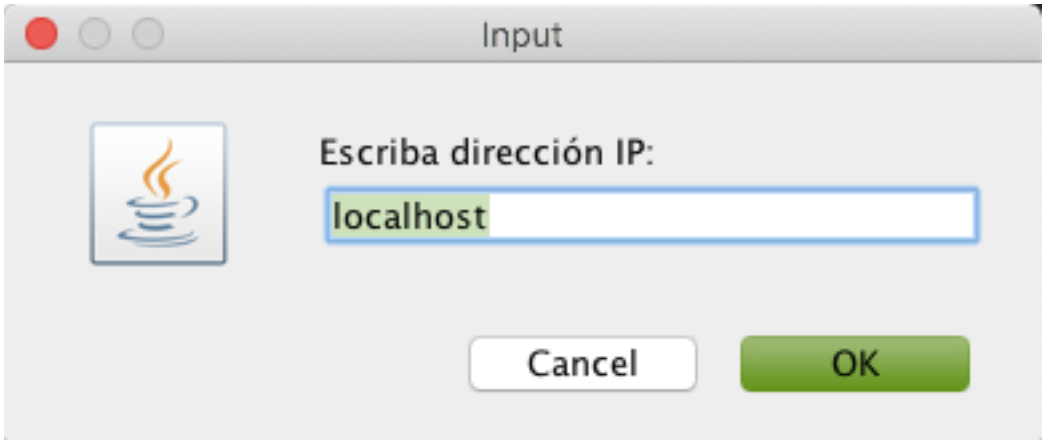
llaves de nuestro HashTable, por lo que procederemos a imprimir el mensaje entrante y responderemos inmediatamente con el valor almacenado para la llave correspondiente. La función está implementada de la siguiente manera:

```
// Método que es llamado cada vez que recibamos un mensaje nuevo.
public void leeRed(Object objeto) {
  // Extraemos el objeto deserializado.
  Mensaje mensaje = (Mensaje) objeto;

  // Verificamos que no sea un mensaje emitido por nosotros mismos.
  if (!mensaje.getUsuario().equalsIgnoreCase(this.usuario + " (I.A.)")
      && !mensaje.getUsuario().equalsIgnoreCase(this.usuario)) {
    // Añadimos el mensaje a nuestra área de mensajes.
    this.areaMensajes.append("\n" + mensaje.getUsuario() + ": " + mensaje.getMensaje());

    // Hacemos un log en la terminal de la ocurrencia.
    System.out.println("Mensaje recibido de " + mensaje.getUsuario() + ": " + mensaje.getMensaje());

    // Verificamos si el mensaje es uno reconocido por nuestra IA.
    if (this.inteligenciaArtificial.containsKey(mensaje.getMensaje().toLowerCase())) {
      // Obtenemos nuestra respuesta del hashtable.
      String respuesta = this.inteligenciaArtificial.get(mensaje.getMensaje());

      // Mandamos la respuesta en red.
      red.escribeRed(new Mensaje(this.usuario + " (I.A.)", respuesta));

      // Añadimos la respuesta a nuestra área de mensajes.
      this.areaMensajes.append("\n" + this.usuario + " (I.A.): " + respuesta);

      // Hacemos un log del mensaje respondido.
      System.out.println("Mensaje respondido por I.A.: " + respuesta);
    }
  }
}
```

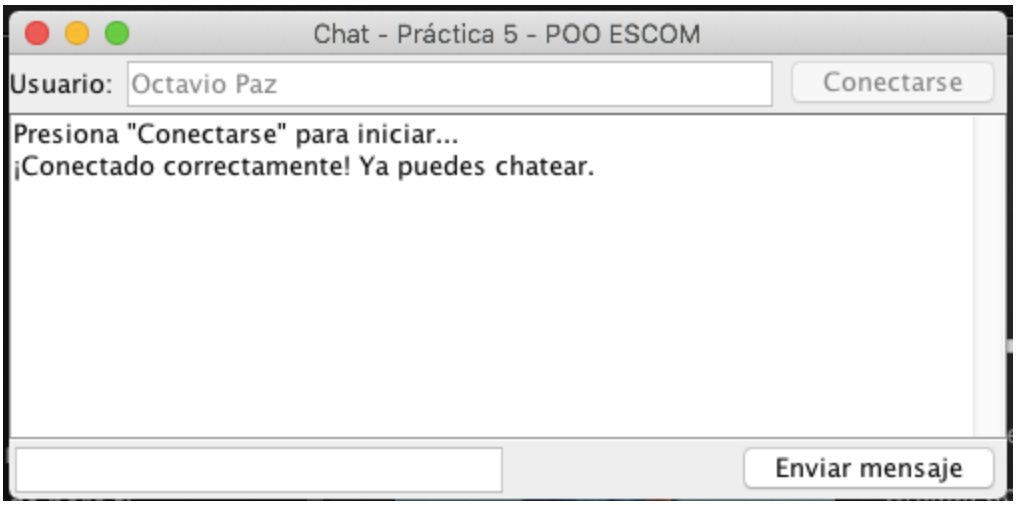
Con estas funciones, es fácil entender el funcionamiento del programa y muestra sus principales características fundamentales. Una ejecución promedio del programa, corriendo el servidor en el fondo, se vería así:



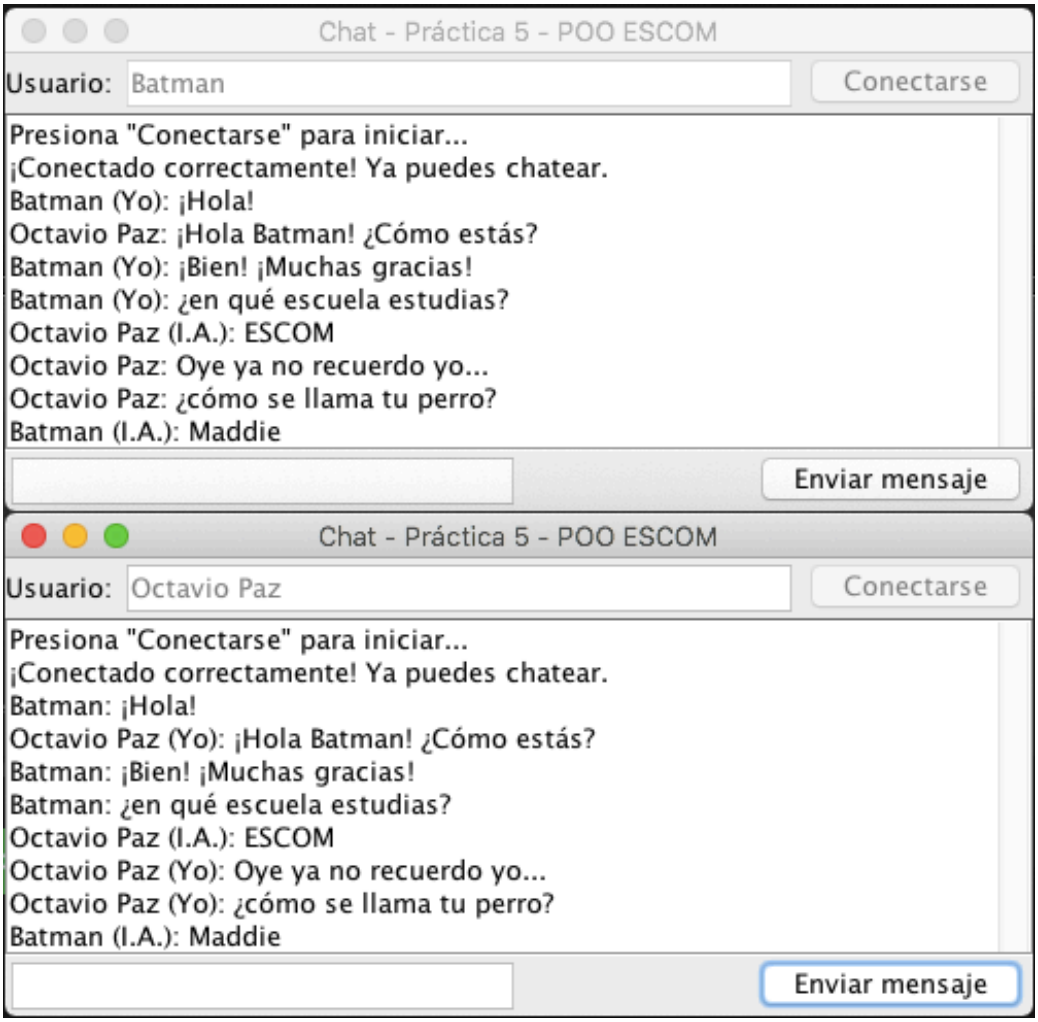
Una vez seleccionado un nombre de usuario, se hace click en “conectarse” que arroja el siguiente mensaje:



Una vez ingresada la dirección IP de nuestro servidor, la interfaz cambiará a esta forma:



Y se puede comenzar con las interacciones:



Conclusión

Si bien el desarrollo de un chatbot siempre conlleva requerimientos técnicos que pueden dar pie a complejidades muy elevadas, sobretodo si se considera el panorama actual de productos y proyectos de software con el único objetivo de mejorar las interacciones entre estos servicios conversacionales con personas, en esta práctica se pudo apreciar de forma cercana las complejidades que se tienen por la parte de interconexión de los clientes para poder lograr una comunicación eficiente.

Es importante que sepamos que dentro de las distintas formas de escribir un Chatbot, lo que se buscaba en esta práctica es tener presente y claro la forma de interactuar de cada uno de nuestros componentes. En este caso, la parte de comunicación en red fue reutilizada de otros proyectos provistos por el profesor, pero no es menos meritorio el trabajo aquí presentado ya que se interactúa directamente con una interfaz gráfica de usuario y se tiene que contemplar distintos escenarios de errores que pueden surgir al ser una aplicación orientada a conexión entre computadoras.

Para los fines prácticos de esta práctica, no se realizó un análisis de desconexiones, reconexiones y demás elementos de un chat “moderno” que permitirían una estabilidad mucho más amplia.

Como siempre, aprendí mucho desarrollando la práctica ya que pude echar mano de conocimientos de otras áreas para integrarlos en un proyecto dónde puedo visualizar distintos elementos computacionales interactuando entre sí.