



**UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA EN INFORMÁTICA**

**LABORATORIO 3
PARADIGMAS DE PROGRAMACIÓN**

Vicente I. Ortiz Arancibia

Profesores: Víctor Flores
Roberto González
Luis Celedón
Fecha de Entrega: 10-08-2018

Santiago de Chile

1 - 2018

TABLA DE CONTENIDOS

TABLA DE ILUSTRACIONES	3
CAPÍTULO 1. INTRODUCCIÓN	4
CAPÍTULO 2. MARCO TEÓRICO.....	5
2.1 PARADIGMA ORIENTADO A OBJETOS	5
2.2. CLASE	5
2.3. OBJETO.....	5
CAPÍTULO 3. DESCRIPCIÓN DEL PROBLEMA.....	5
3.1 ANÁLISIS	7
CAPÍTULO 4. DESCRIPCIÓN DE LA SOLUCIÓN	9
CAPÍTULO 5. RESULTADOS	13
CAPÍTULO 6. CONCLUSIONES	16
BIBLIOGRAFÍA	16

TABLA DE ILUSTRACIONES

Ilustración 1 .-Diagrama UML de análisis	8
Ilustración 2 .- Diagrama conversacional chatbot.....	8
Ilustración 3.- Ejemplo de arreglo de Strings que muestra las palabras posibles a identificar, y las posibles respuestas.....	9
Ilustración 4.- Extracto del método respuesta de la clase Chatbot.	10
Ilustración 5.- Método crearLog de clase Mensaje.	11
Ilustración 6.- Método leerLog de clase Mensaje.	12
Ilustración 7.- Mensaje de bienvenida	13
Ilustración 8.- Uso de comando !beginDialog	13
Ilustración 9.- Envío de mensajes.....	13
Ilustración 10.- Uso de comando !endDialog	13
Ilustración 11.- Uso del comando !saveLog.....	14
Ilustración 12.- Archivo creado por el comando !saveLog	14
Ilustración 13.- Diagrama UML Final	15

CAPÍTULO 1. INTRODUCCIÓN

Un paradigma es una teoría o conjunto de teorías cuyo núcleo central se acepta sin cuestionar y que suministra la base y modelo para resolver problemas y avanzar en el conocimiento. En el contexto informático, se dice que un paradigma es un estilo de desarrollo de programas. Es decir, un modelo para resolver problemas computacionales.

Durante este semestre se estudiarán 4 de estos paradigmas, el funcional, lógico, orientado a objetos e imperativo que son los principales paradigmas existentes actuales.

El presente informe se desarrollará en un programa bajo el paradigma orientado a objetos, que ve un programa como un conjunto de clases. Las clases son entidades que combinan atributos, que, si lo comparamos que paradigmas como el imperativo serían sus variables, y un método, que haciendo la misma analogía serían funciones. El POO surge para solventar los problemas que planteaban otros paradigmas, como el imperativo, con el objeto de elaborar programas y módulos más fáciles de escribir, mantener y reutilizar.

El objetivo principal de este informe es desarrollar un chatbot (ver anexo para la definición), cuyo tema en este caso es un asistente de vendedor de preservativos. Este programa emulara de una manera muy básica la inteligencia artificial que manejan actualmente chatbots muy avanzados, y que se han atrevido a desafiar el test de Turing. También se busca demostrar y aplicar los conocimientos adquiridos en cátedra y laboratorio sobre el paradigma orientado a objetos y el lenguaje de programación Java.

CAPÍTULO 2. MARCO TEÓRICO

2.1 PARADIGMA ORIENTADO A OBJETOS

La programación orientada a objetos (POO) nace de la necesidad de encontrar un modelo de programación que acelerase la productividad de los desarrolladores de *software*. Para ello, el concepto clave es la reutilización, a través de la herencia y la asociación dinámica. Por otro lado, la complejidad de los programas se hacía cada vez mayor, por lo que existía la necesidad de abstraerse de ciertos conceptos y concentrarse en resolver el problema, lo que derivó en la idea de tipos de datos abstractos.

2.2. CLASE

Un tipo de dato abstracto puede definirse a través de una clase. Una clase relaciona una serie de atributos (variables) y una serie métodos (funciones u operaciones) que operan sobre la clase.

2.3. OBJETO

Es una unidad compacta, que se encuentra en tiempo y que hace las tareas dentro de un programa. En programación, los objetos se utilizan para el modelamiento de entidades y/o de objetos en el mundo real (el objeto lápiz, goma, mesa, etc.). En otras palabras, un objeto es la representación de un concepto dentro del programa, y tiene la información que necesita para su correcta abstracción, es decir, datos que permiten describir sus características (atributos) como sus operaciones que pueden realizarse sobre esta.

CAPÍTULO 3. DESCRIPCIÓN DEL PROBLEMA

Se solicita desarrollar un programa en el lenguaje de programación Java que simule de la mejor manera un chatbot. Este chatbot debe tener características mínimas como:

- Tener un contexto y temática definida.
- Diseño de flujos conversaciones
- Considerar un amplio abanico de respuestas, con el fin de cautivar mediante una conversación activa con el usuario.
- Incorporar respuestas que permitan reaccionar a distintas situaciones.
- Debe responder a mensajes cuya estructura gramatical sea simple, en español y conocido por el chatbot.
- Ante entradas desconocidas, el chatbot debe contar con un repertorio de respuestas que le permitan redirigir la conversación con el usuario.

Además, como parte del diseño orientado a objetos de la solución el enunciado, se debe considerar como mínimo cubrir las siguientes entidades a través de múltiples clases relacionadas:

1. **Chatbot:** Corresponde a la(s) entidad(es) que interactuarán con el usuario respondiendo sus consultas. El chatbot deberá tener personalidades (ej.: coloquial, formal, agresivo, etc.) determinado por un Seed (semilla pseudoaleatoria) para lograr variabilidad de las respuestas del chatbot.
2. **Diseñar un log:** Está formado por los mensajes que emite tanto el chatbot como el usuario. Cada entrada del Log deberá incluir el mensaje, la fecha en que se emitió (timestamp) e indicar su emisor (usuario o chatbot) y deberán ser almacenadas para futura referencia.
3. **Usuario:** Esta entidad está formada por toda la información útil que pueda obtener el chatbot sobre el usuario a medida que conversa con éste.

Finalmente, debe tener las siguientes interacciones mínimas con el chatbot que serán ingresadas con el símbolo "!". Cualquier diálogo que no empiece con "!", debe ser considerado una interacción directa con el chatbot.

1. **!endDialog** : Termina la conversación con el chatbot, entregando éste el último mensaje de despedida.
2. **!beginDialog seed**: Permite iniciar el chatbot con un valor semilla (*seed*) para su personalidad si se omite el valor semilla, se cargará una personalidad predeterminada (default). Si ya se estaba ejecutando una conversación, se deberá terminar esta y luego reiniciar la conversación con el nuevo chatbot.
3. **!saveLog**: Escribe un archivo de texto plano el log completo de la conversación actual (incluyendo marcas de tiempo e información relevante del chatbot). El archivo será guardado en el directorio en que se ejecuta el programa y su formato del nombre de archivo deberá basarse en la marca de tiempo en la que se envió el comando. Ej.: 2018-05-30_15-45.log
4. **!loadLog nombreArchivo**: Lee desde un archivo de texto plano generado por !saveLog una conversación de nombre *nombreArchivo* y permitirá continuarla desde dicho punto.
5. **!rate notaChatbot notaUsuario**: Permite evaluar el desempeño del chatbot (y del usuario) tras haber terminado una conversación. Las evaluaciones serán entre 1 y 5, o 0 si no se puede determinar. Dichos puntajes deberán almacenarse con su marca de tiempo.

3.1 ANÁLISIS

Como se mencionó anteriormente, una de las ventajas que nos ofrece el paradigma orientado a objetos, es el hecho de que las clases nos permite crear una modularización de la solución que implementaremos, por lo que podemos separar el problema, y luego la solución, en distintas entidades que trabajaran por si solas, pero que se juntarán en algún momento, permitiendo que el programa funcione de manera óptima.

En primer lugar, se debe analizar los aspectos que tienen que ver meramente con el chatbot, y los actores que lo incluyen. A grandes rasgos, podemos identificar 3 grandes actores dentro del flujo conversacional que implica un chatbot, el usuario no-maquina, que sería el que interactúa con la maquina; el usuario-maquina (chatbot), que almacena todas las respuestas configuradas a dar y que puede procesar el mensaje del usuario no-máquina para dar una respuesta acorde; y el mensaje, que sería el actor encargado almacenar toda la información del flujo conversacional.

Si esto lo llevamos a implementación de POO, podríamos decir que dentro de nuestra solución existirán 3 grandes clases, la clase Usuario, la clase Chatbot y la clase Mensaje.

El usuario podrá almacenar toda la información importante que se necesita almacenar respecto a su participación dentro del flujo conversacional, como el ID, el nombre, etc.

El chatbot almacenará las respuestas preconfiguradas que tiene para dar al usuario no-maquina, teniendo un amplio abanico de respuestas en función de la personalidad que este puede tomar. Además el chatbot será capaz analizar el mensaje de entrada del usuario, y procesar internamente cual será la mejor respuesta al mensaje ingresado por el usuario.

Finalmente, el mensaje es el encargado de almacenar todo el flujo conversacional entre el usuario y el chatbot, incluyendo datos importantes como la hora, las evaluaciones que se le pueden hacer al chatbot, entre otra información importante.

Si lo lleváramos tempranamente a un diagrama UML, se vería algo así:

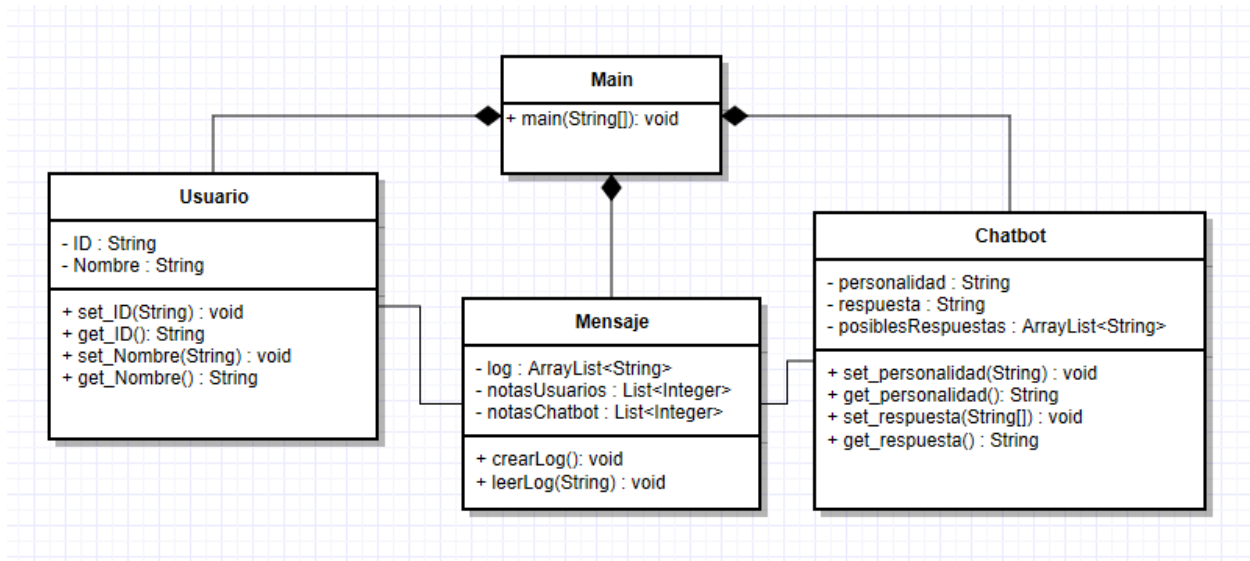


Ilustración 1 .-Diagrama UML de análisis

Ahora, además de identificar las clases existentes en la solución que se presentará, otro gran desafío es el chatbot funcionando en sí. Ya que hay que crear un método que sea capaz de identificar la 'entrada' que nos da el usuario, evaluarla, y dar una respuesta acorde a lo que el usuario este solicitando, además de tratar de ser lo menos 'robot' posible en la respuesta. Para esto, se deberá diseñar un diagrama conversacional que nos permita ordenar las posibles entradas de los usuarios y diseñar las respuestas que el chatbot tiene que ser capaz de dar.

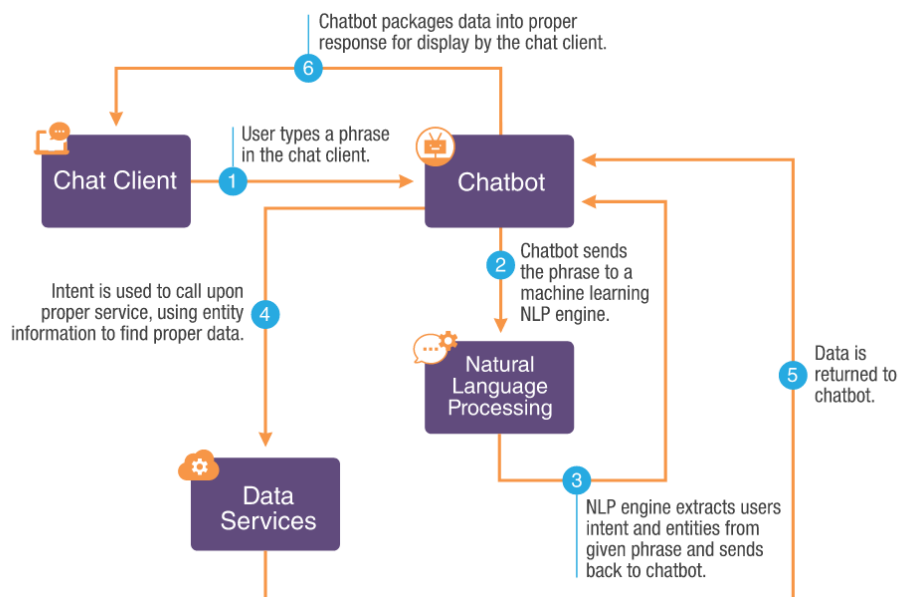


Ilustración 2 .- Diagrama conversacional chatbot

CAPÍTULO 4. DESCRIPCIÓN DE LA SOLUCIÓN

Como se mencionó en el análisis del problema, se crearon 3 grandes clases en el proyecto para que el programe funcione, a continuación, se procederá a explicar una por una estas clases.

4.1. CLASE USUARIO

En esta clase se crearon atributos que están asociados a datos importantes del usuario dentro del flujo conversacional, como lo son el ID y el nombre. Además se crearon métodos getter y setter, para poder modificar los atributos de esta clase.

4.2. CLASE CHATBOT

En esta clase finalmente se tuvieron que implementar más atributos de los que se diseñaron en la etapa de análisis. Ya que además de los atributos y métodos que se tenían contemplados, se tuvieron que agregar atributos *String[]* que contuvieran las posibles respuestas del Chatbot a determinado mensaje, además de las palabras que es posible de analizar e identificar el chatbot.

Por ejemplo, si el usuario escribiera por consola “me gustaría encargar 36 preservativos”, el chatbot entraría al arreglo de *Strings* **confirmarCompra** buscando las posibles palabras que puede identificar, y al identificar la palabra “encargar”, entraría al arreglo de *Strings* **RCCompra** para entregar una de las respuestas almacenadas en ese arreglo.

```
private String[] confirmarCompra = {"necesito", "encargar", "encargar?"};  
private String[] RCCompra = {"Perfecto, dejeme confirmar si hay stock disponible",  
                             "Deja revisar si me quedan.",  
                             "RESPUESTA INDEFINIDA 3"};
```

Ilustración 3.- Ejemplo de arreglo de Strings que muestra las palabras posibles a identificar, y las posibles respuestas.

En el párrafo anterior, se mencionó implícitamente el proceso que se creó en el método **respuesta**, que es el encargado de analizar la entrada del Usuario, y entregar una respuesta acorde a lo ingresado. Para esto, en la presente clase, como se mencionó se creó este método que a continuación se muestra un extracto:

```

public void respuesta(String[] oracion) {
    int largo = oracion.length;
    int aux = 0;
    date = new Date();

    for(int i=0; i<largo; i++) {
        for(int j=0; j<informacion.length; j++) {
            if(oracion[i].equals(informacion[j])) {
                answer = hourdateFormat.format(date) + "Chatbot> " + Rinformacion[intPersonalidad];
                aux = 1;
            }
        }
    }
}

```

Ilustración 4.- Extracto del método respuesta de la clase Chatbot.

Como se aprecia, el único parámetro de este método es un arreglo de *String* llamado *oración*. Este parámetro es el mensaje ingresado por el usuario guardado en un arreglo de *Strings* que separa la oración en un espacio. Por lo que si el usuario ingreso “me gustaría encargar 36 preservativos”, el parámetro sería el siguiente arreglo: [“me”, “gustaría”, “encargar”, “36”, “preservativos”]. Este método analiza todos los elementos del arreglo hasta identificar alguna coincidencia con la “base” que existe preconfigurada dentro del chatbot. Si no encuentra nada, le avisa al usuario que no ha entendido su mensaje. Al encontrar una coincidencia, se procede a guardar la respuesta en la variable auxiliar *answer* tomando en cuenta la personalidad adoptada aleatoriamente por el chatbot, guardada en el atributo *intPersonalidad*.

Además de tener el método que hace la labor principal de un chatbot, esta clase también almacena los métodos **beginDialog** y **endDialog**, que son los encargados de dar respuesta a los input del usuario “!beginDialog” y “!endDialog”. Estos dan el mensaje de bienvenida y despedida, además de, en el caso de beginDialog entregar el ID al chat, y en conjunto dejar las marcas de tiempo donde se inicializo y finalizó el flujo conversacional.

4.3. CLASE MENSAJE

La clase mensaje, es la que cumple la función de almacenar todos los mensajes e interacciones hechas entre usuario y chatbot. Se creó el atributo **log**, que es del tipo *ArrayList<String>*, ya que almacena todas las líneas de texto mostradas por consola en una arreglo de strings, que se usará en caso de que el usuario utilice la opción por consola “!saveLog”.

Además esta clase contiene los atributos **horaRate**, **notasUsuario** y **notasChatbot**, que son atributos de tipo *ArrayList<String>* destinados a guardar las notas, en el caso de **notasUsuario** y **notasChatbot**, y a guardar la hora de la evaluación, en el caso de **horaRate**, entregada por consola con relación al chatbot activo en el momento.

Los principales métodos que contiene esta clase es **crearLog** y **leerLog**.

En el caso del primero, **crearLog** es el encargado de ejecutar el comando “!saveLog”, y procede a utilizar el atributo **log** creado automáticamente por la clase mensaje, y guarda línea por línea lo almacenado en **log** en un archivo de texto de nombre creado en función a la hora de la conversación, con extensión .log, la conversación o conversaciones, efectuada entre el usuario y el chatbot.

```
public void crearLog(){
    Date date = new Date();
    DateFormat fileFormat = new SimpleDateFormat("yyyy-MM-dd_HH-mm");
    String fileName = "C:\\Users\\vicen\\Desktop\\" +fileFormat.format(date)+ ".log";
    FileWriter fichero = null;
    PrintWriter pw = null;
    try
    {
        fichero = new FileWriter(fileName);
        pw = new PrintWriter(fichero);
        for(int i=0; i<log.size(); i++) {
            pw.println(log.get(i));
        }
        for(int i=0; i<horaRate.size();i++) {
            pw.println(horaRate.get(i) + " Usuario:" + notasUsuario.get(i) + " Chatbot:" + notasChatbot.get(i));
        }
    }catch(Exception e) {
        e.printStackTrace();
    }finally {
        try {
            if(null != fichero) {
                fichero.close();
            }
        }catch(Exception e2) {
            e2.printStackTrace();
        }
    }
}
```

Ilustración 5.- Método crearLog de clase Mensaje.

Por otro lado, **leerLog**, es el encargado de ejecutar el comando “!loadLog *NombreArchivo*”. Este método se encarga de leer un archivo de texto “*NombreArchivo*” que contiene una conversación, y desplegarlo por consola. Además deja el log abierto para que pueda seguir añadiendo nuevas conversaciones al flujo conversacional mostrado por pantalla.

```

public void leerLog(String nombreArchivo) {
    log.clear();
    File archivo = null;
    FileReader fr = null;
    BufferedReader br = null;
    String fileName = "C:\\Users\\viken\\Desktop\\" + nombreArchivo;
    try {
        // Apertura del fichero y creacion de BufferedReader para poder
        // hacer una lectura comoda (disponer del metodo readLine()).
        archivo = new File (fileName);
        fr = new FileReader (archivo);
        br = new BufferedReader(fr);
        // Lectura del fichero
        String linea;
        while((linea=br.readLine())!=null) {
            // System.out.println("1");
            log.add(linea);
            System.out.println(linea);
            //lineaAux = linea;
        }
    }
    catch(Exception e){
        e.printStackTrace();
    }finally{
        // En el finally cerramos el fichero, para asegurarnos
        // que se cierra tanto si todo va bien como si salta
        // una excepcion.
        try{
            if( null != fr ){
                fr.close();
            }
        }catch (Exception e2){
            e2.printStackTrace();
        }
    }
}

```

Ilustración 6.- Método leerLog de clase Mensaje.

4.4. MAIN

La clase main es la encargada de inicializar todas las clases como objetos para que el programa pueda funcionar. Además es la encargada en primera instancia de inicializar el chatbot por consola, y reconocer los comandos que puede ingresar el usuario, como: !beginDialog, !endDialog, !saveLog, entre otros. En caso de no ser un comando reconocido, el main procede a revisar si identifica alguna palabra dentro de la oración que el usuario ingreso, para poder dar una respuesta satisfactoria. En caso de no reconocer ni el comando, ni una oración válida para el chatbot, procede a avisarle al usuario que no reconoció lo que ingreso por consola.

CAPÍTULO 5. RESULTADOS

En relación con los requerimientos mínimos funcionales, se puede mostrar a continuación que se alcanzó un 100% de desarrollo en cada uno de los puntos mencionados en la descripción del problema.

Al inicializar el programa, sale automáticamente el mensaje de bienvenida del chatbot, además de la línea de comando lista para ser usada:

```
¡Bienvenido al asistente virtual de Vo'Confia!  
- Aquí podrás conversar con uno de nuestros ejecutivos para tener información respecto a los preservativos que vendemos  
- Esperamos tengas una buena experiencia, ¡que tengas un buen día!  
  
[10/08/2018 02:53:07] Usuario>
```

Ilustración 7.- Mensaje de bienvenida

Podemos inicializar la conversación con el comando **!beginDialog**, que hará que el chatbot de un mensaje de bienvenida al chat concretamente, asignará un ID al Usuario para poder ligarlo con la conversación, además de tomar una personalidad según el parámetro *seed* que debe entregar el usuario:

```
[10/08/2018 02:53:07] Usuario> !beginDialog 312  
[USER ID: 20180210080753 | PERSONALIDAD: 2]  
[10/08/2018 02:57:15] Chatbot> Hola! como te puedo ayudar el día de hoy?  
[10/08/2018 02:57:15] Usuario>
```

Ilustración 8.- Uso de comando !beginDialog

A continuación podemos proceder a enviar un mensaje, o si queremos, finalizar la conversación, pero con fin de poder explicar mejor los resultados, procederemos a enviar un mensaje, y luego finalizaremos la conversación.

```
[10/08/2018 03:01:46] Usuario> Hola! me gustaría informacion  
[10/08/2018 03:01:59] Chatbot> Estimado, le invito a revisar el fanpage, ya que está toda la información publicada ahí.  
[10/08/2018 03:01:59] Usuario>
```

Ilustración 9.- Envío de mensajes

Luego de obtener todo lo que el usuario requiere del chatbot, este puede proceder a finalizar la conversación con el comando **!endDialog**, finalizando así la conversación actual, brindándole la hora de termino y un mensaje de despedida.

```
[10/08/2018 03:01:59] Usuario> !endDialog  
[10/08/2018 03:04:53] Chatbot> Gracias por preferir Vo'Confia, hasta pronto! #SexualidadResponsable  
[10/08/2018 03:04:53] Usuario>
```

Ilustración 10.- Uso de comando !endDialog

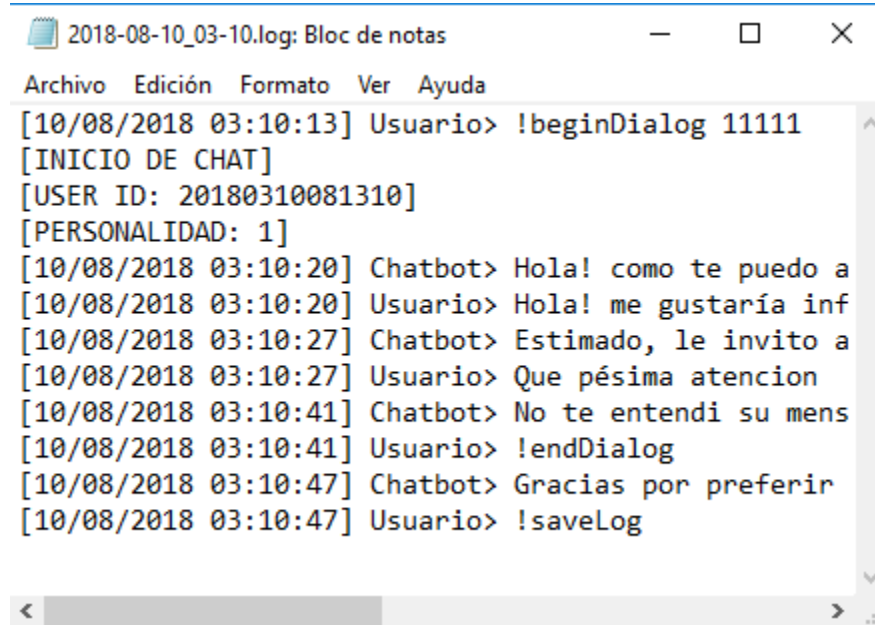
Como se ve en la figura anterior, la línea de comando sigue abierta, ya que el usuario puede seguir interactuando con el programa, ya sea iniciando una nueva conversación, o también tiene la opción de utilizar los comandos **!saveLog**, **!loadLog**, **!rate** y **!chatbotPerformance**, cuyas funcionalidades por pantalla son explicadas más extensamente en el Manual de Usuario.

A continuación se mostrará cómo funciona el comando **!saveLog**:

```
[10/08/2018 03:10:13] Usuario> !beginDialog 11111
[USER ID: 20180310081310 | PERSONALIDAD: 1]
[10/08/2018 03:10:20] Chatbot> Hola! como te puedo ayudar el día de hoy?
[10/08/2018 03:10:20] Usuario> Hola! me gustaría informacion
[10/08/2018 03:10:27] Chatbot> Estimado, le invito a revisar el fanpage, ya que está toda la información publicada ahí.
[10/08/2018 03:10:27] Usuario> Que pésima atencion
[10/08/2018 03:10:41] Chatbot> No te entendí su mensaje.
[10/08/2018 03:10:41] Usuario> !endDialog
[10/08/2018 03:10:47] Chatbot> Gracias por preferir Vo'Confia, hasta pronto! #SexualidadResponsable
[10/08/2018 03:10:47] Usuario> !saveLog
[10/08/2018 03:10:53] Usuario>
```

Ilustración 11.- Uso del comando !saveLog

Como se explicó anteriormente, **!saveLog** guarda el historial de conversaciones realizados en el programa, y los archiva en un archivo de texto .log, que se nombra en relación directa con la hora del programa, y al usarlo, en el archivo de texto queda como resultado lo siguiente:



```
2018-08-10_03-10.log: Bloc de notas
Archivo Edición Formato Ver Ayuda
[10/08/2018 03:10:13] Usuario> !beginDialog 11111
[INICIO DE CHAT]
[USER ID: 20180310081310]
[PERSONALIDAD: 1]
[10/08/2018 03:10:20] Chatbot> Hola! como te puedo a
[10/08/2018 03:10:20] Usuario> Hola! me gustaría inf
[10/08/2018 03:10:27] Chatbot> Estimado, le invito a
[10/08/2018 03:10:27] Usuario> Que pésima atencion
[10/08/2018 03:10:41] Chatbot> No te entendí su mens
[10/08/2018 03:10:41] Usuario> !endDialog
[10/08/2018 03:10:47] Chatbot> Gracias por preferir
[10/08/2018 03:10:47] Usuario> !saveLog
```

Ilustración 12.- Archivo creado por el comando !saveLog

Además de los resultados funcionales que se acaban de exponer, el resultado final del Diagrama UML fue el siguiente:

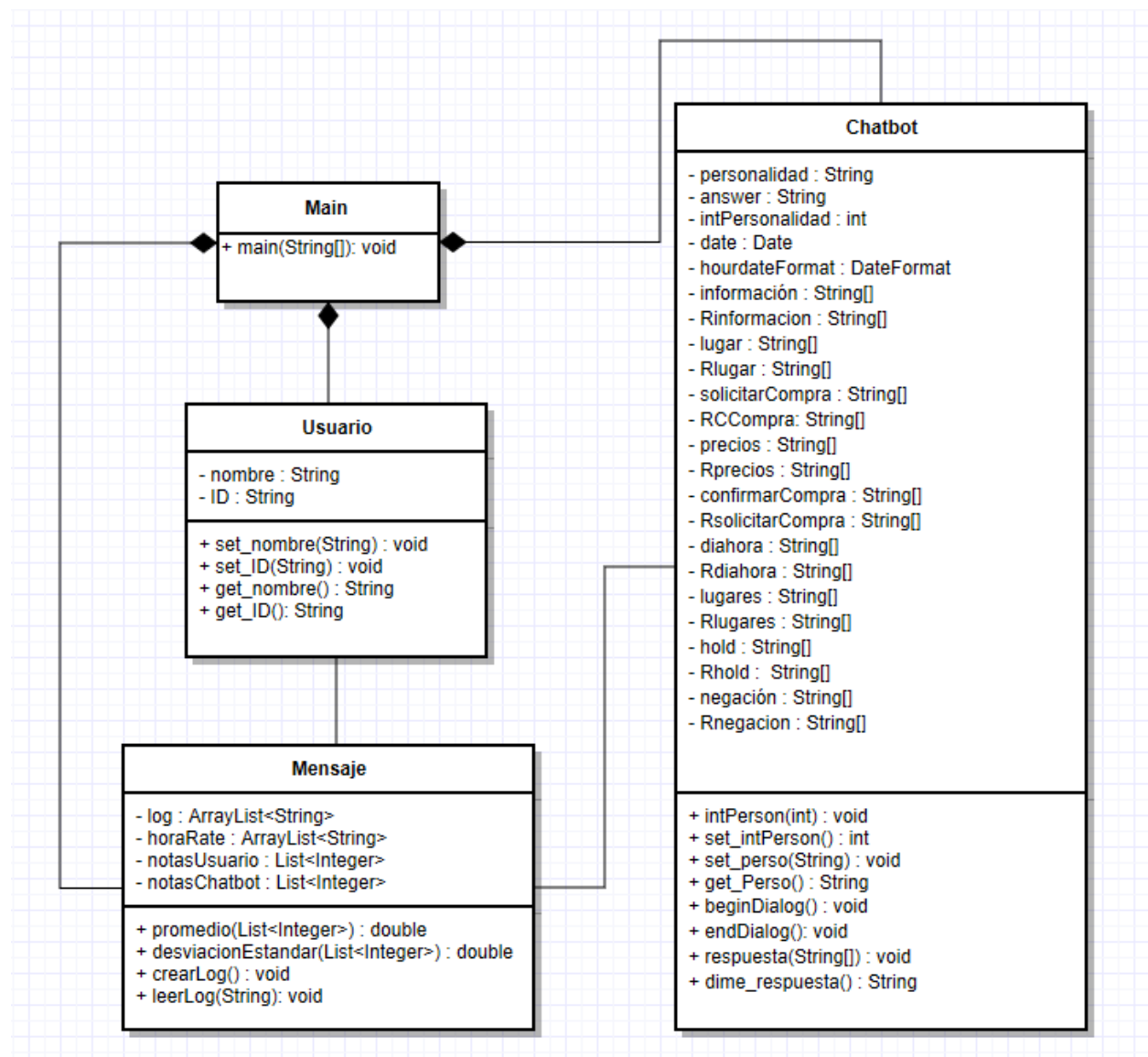


Ilustración 13.- Diagrama UML Final

CAPÍTULO 6. CONCLUSIONES

Dentro de los paradigmas que se estudian en el semestre, el orientado a objetos es quizás el que hace el trabajo más “fácil” el trabajo para un proyecto como este. Ya que los actores dentro de un flujo conversacional están muy bien definidos, por lo que trabajar con la modularización que ofrece trabajar con POO, hace que todo sea más ordenado.

Como lenguaje, JAVA debe ser de los lenguajes de programación más populares actualmente, por lo que encontrar documentación sobre todo es un trabajo medianamente fácil, el problema es que como popular que es, también tiene muchos conceptos que quizás no se alcanzan a utilizar como herramientas útiles dentro de los proyectos que uno desarrolla. Por ejemplo, para este laboratorio, por temas de tiempo, no se pudo aplicar lo que recientemente se aprendió en teoría como lo son los conceptos de dependencias, herencias, acoplamiento, entre otros. Pero si se ocuparon otras técnicas como las relaciones, los objetos, etc.

En comparación a la forma de trabajar con paradigmas anteriores, quizás fue levemente más fácil justamente por lo mencionado anteriormente, que es muy fácil encontrar documentación y apoyo al respecto de JAVA puntualmente, y además que POO hace que este proyecto se haga de manera muy óptima.

Finalmente, se puede afirmar que se desarrolló el chatbot satisfactoriamente y cumpliendo con todos los requerimientos funcionales, además, se puede decir que se ha cumplido a cabalidad el objetivo de este laboratorio que era aplicar y demostrar el conocimiento que se tiene del paradigma orientado a objetos y del lenguaje JAVA..

BIBLIOGRAFÍA

- Enunciado laboratorios 1 – 2018. Paradigmas de programación. Profesores varios.
- http://aprendeonline.udea.edu.co/lms/men_udea/mod/page/view.php?id=19537