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



**LABORATORIO 3**

**PARADIGMAS DE PROGRAMACIÓN**

Gabriel Gaete L.

|  |  |
| --- | --- |
| Profesores: | Roberto González |
|  | Daniel Gacitúa |
|  | Víctor Flores |
| Fecha de Entrega: | 10 de Agosto del 2018 |

Santiago de Chile

1 - 2018

# TABLA DE CONTENIDO

[TABLA DE FIGURAS 2](#_Toc521341825)

[CAPÍTULO 1. INTRODUCCIÓN 3](#_Toc521341826)

[CAPÍTULO 2. MARCO TEÓRICO 4](#_Toc521341827)

[2.1 PARADIGMA ORIENTADO A OBJETOS 4](#_Toc521341828)

[2.2 CLASES 4](#_Toc521341829)

[2.3 OBJETO 4](#_Toc521341831)

[CAPÍTULO 3. DESCRIPCIÓN DEL PROBLEMA 5](#_Toc521341832)

[3.1 ANÁLISIS 5](#_Toc521341833)

[CAPÍTULO 4. DESCRIPCIÓN DE LA SOLUCIÓN 6](#_Toc521341834)

[CAPÍTULO 5. RESULTADOS 12](#_Toc521341835)

[CAPÍTULO 6. CONCLUSIONES 15](#_Toc521341836)

[CAPÍTULO 7. REFERENCIAS 16](#_Toc521341837)

# TABLA DE FIGURAS

[Figura 1 Diagrama de Clases 7](#_Toc521683657)

[Figura 2 Representación de un mensaje 8](#_Toc521683658)

[Figura 3 Método para intersectar mensajes del usuario 9](#_Toc521683659)

[Figura 4 Método que permite al Chatbot dar la bienvenida al Usuario. 9](#_Toc521683660)

[Figura 5 Código fuente de la clase Log 10](#_Toc521683661)

[Figura 6 Código fuente de la clase Main 10](#_Toc521683662)

[Figura 7 Implementación instrucción !rate 11](#_Toc521683663)

[Figura 8 Implementación instrucción !saveLog 11](#_Toc521683664)

[Figura 9 Implementación instrucción !beginDialog seed 12](#_Toc521683665)

[Figura 10 Implementación !endDialog 12](#_Toc521683666)

[Figura 11 Ejemplo de beginDialog 13](#_Toc521683667)

[Figura 12 Ejemplo de intercambio de mensajes 13](#_Toc521683668)

[Figura 13 Ejemplo de !saveLog 14](#_Toc521683669)

[Figura 14 Registro dentro del log 14](#_Toc521683670)

[Figura 15 Ejemplo de !endDialog y !exit 15](#_Toc521683671)

# CAPÍTULO 1. INTRODUCCIÓN

Un *chatbot* o un bot conversacional es *“un robot capaz de hablar e interactuar imitando el comportamiento humano, ya sea oral o por escrito, respondiendo a las preguntas y reclamos de los usuarios.”* (Herrero, s.f.)En otras palabras, es un programa que permite simular una conversación con una persona, entregando respuestas automáticas a entradas hechas por un usuario. Esta conversación, habitualmente se establece a través de texto, sin embargo, hay modelos que disponen de una interfaz multimedia. También se han desarrollado chatbots que utilizan conversores de texto a sonido, dotando de mayor realismo a la interacción con el usuario.

El presente informe tiene por objetivo principal ser una referencia a una línea de pensamiento y a un contexto de desarrollo del código fuente que lo acompaña para la presentación del segundo laboratorio del curso *“Paradigmas de programación”*. En esta oportunidad, se hará uso del paradigma de orientación a objetos bajo el lenguaje de programación *Java*.

El paradigma de orientación a objetos (*Object-oriented Programming)* corresponde a un paradigma basado en el concepto de “objetos” que contienen información, conocida comúnmente como *atributos,* y que también son capaces de realizar ciertas acciones u operaciones que son propias de este objeto, las cuales se conocen como *métodos.*

El problema a resolver en este laboratorio es el desarrollo de un *chatbot*, con el cual se deberá mantener una conversación básica, protocolar, en la que exista un flujo conversacional coherente. El contexto para este chatbot será una venta de pasajes a capitales regionales dentro de Chile.

La solución implementada trabaja en base a un conjunto de clases que definen el comportamiento de los elementos que interactúan en el programa, tales como el Chatbot, un usuario, el chat propiamente tal, los mensajes intercambiados, y el registro de mensajes que se han intercambiado (es decir, un *log*).

Se establece entonces, como objetivo en este laboratorio, el aplicar y demostrar los conocimientos adquiridos en cátedra con respecto a lo que es el paradigma de programación orientado a objetos, utilizando el lenguaje de progamación *Java*.

# CAPÍTULO 2. MARCO TEÓRICO

## 2.1 PARADIGMA ORIENTADO A OBJETOS

El paradigma orientado a objetos es una forma de construir, diseñar e implementar programas basados en comunidades de objetos que combinan estados, comportamientos e identidades. El estado está compuesto por datos, también llamados atributos; el comportamiento está definido por una serie de acciones que se realizan y son los llamados métodos, mientras que la identidad es lo que diferencia a un objeto del resto de los objetos dentro de esta comunidad.

Los programas bajo este paradigma se basan en la relación y la colaboración que se pueda establecer entre objetos. La comunicación entre los diferentes objetos se da mediante “mensajes” entre ellos. Estos mensajes son llevados a cabo mediante el llamado a métodos de un cierto objeto.

## 2.2 CLASES

## Es una definición de un objeto. Es un contenedor de uno o más datos, junto con las propiedades que lo manipulan llamados *“métodos”.* En otras palabras, se puede decir que una clase es el modelo a través del cual se generarán los objetos dentro del programa. Como este modelo, que describe las propiedades es un conjunto de variables (atributos) y métodos que pueden ser utilizados como objetos en cualquier punto del programa, es que las clases son uno de los pilares fundamentales de la programación bajo el paradigma de orientación a objetos.

## 2.3 OBJETO

Es una unidad compacta, que se encuentra en tiempo de ejecución y que realiza las tareas propias de un programa. Los objetos en programación se utilizan para modelar objetos o entidades del mundo real (el objeto perro, gato, o tienda, por ejemplo). Un objeto es, en otras palabras, la representación dentro de un programa de un concepto, y contiene toda la información necesaria para abstraerlo: datos que permiten describir sus atributos y operaciones que pueden realizarse sobre los mismos.

De forma rápida, podría decirse también que un objeto es una *instancia* de una clase.

# CAPÍTULO 3. DESCRIPCIÓN DEL PROBLEMA

Se solicita crear un programa en el lenguaje de programación *Java* que simule un bot conversacional (chatbot). Para este laboratorio, el tema del chatbot será una venta de pasajes a capitales regionales de Chile. Este debe funcionar en base a ciertas clases/estructuras como lo son un chatbot, un usuario, y un *log* o historial de conversaciones. El programa debe ser capaz de realizar las siguientes funcionalidades:

1. **!beginDialog seed:** Esta funcionalidad 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 ésta y luego reiniciar la conversación con el nuevo chatbot.
2. **!endDialog:** Ésta funcionalidad termina la conversación con el chatbot, entregando éste último un mensaje de despedida.
3. **!saveLog:** Escribe en 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 se basará en la marca de tiempo en la que se envió el comando. Ej: 2018-05-30\_15-45.log
4. **!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.

Estas funcionalidades tienen un formato definido para su implementación y desarrollo, trabajan con estructuras específicas (principalmente listas). Se tiene la libertad de definir e implementar éstas estructuras acorde a las necesidades de la implementación deseada.

## 3.1 ANÁLISIS

El principal problema que se presenta en este laboratorio radica en cómo definir las clases necesarias para la implementación de la solución. A primera vista, son necesarias 5 clases (Usuario, Chatbot, Log, Mensaje y Chat).

# CAPÍTULO 4. DESCRIPCIÓN DE LA SOLUCIÓN

Para llevar a cabo este programa, se han utilizado 6 clases propias y un conjunto de clases de Java, principalmente *ArrayList* y *Scanner,* para lo que es el manejo de las respuestas que es capaz de entregar el chatbot. La clase principal del programa corresponde a Chat, siendo aquí donde se instancian los demás objetos, como los participantes de este chat, con su respectivo historial de mensajes. En esta misma clase, es donde se realiza la distinción en lo que ha dicho el Usuario, es decir, si es que el Usuario ha ingresado una instrucción especial, o de lo contrario, si quiere intercambiar un mensaje con el bot.

Debido al gran tamaño, se ha decidido intencionalmente no incluir el código fuente dentro de éste documento.

La siguiente figura, permite reflejar las clases creadas y la relación que mantienen entre ellas. De esta forma, fue implementada la solución al problema.

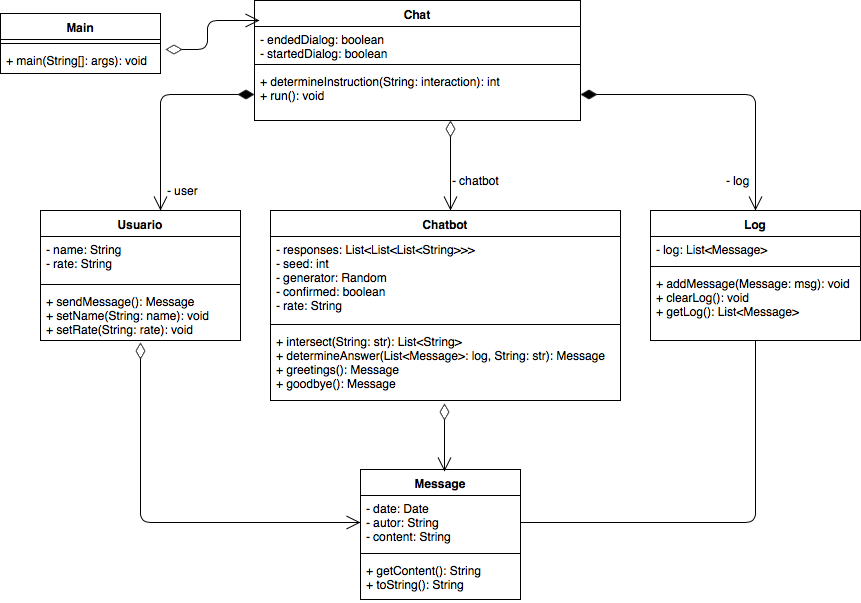


Figura 1 Diagrama de Clases

Antes de entrar de lleno en lo que son las clases requeridas por el enunciado, se debe saber que tanto la clase Usuario, como la clase Chatbot y la clase Log, trabajan en base a la clase Mensaje (o *Message*). Esta clase *Message* permite mantener un mayor orden y claridad a la hora de almacenar los mensajes dentro del registro. En ésta clase *Message*, se establece un nombre o remitente del mensaje, una fecha de envío, y el contenido del mensaje. Su método más importante es *toString()*, el cual permite obtener los datos del mensaje siguiendo un patrón establecido. La siguiente figura, muestra el código fuente de la clase *Message*.

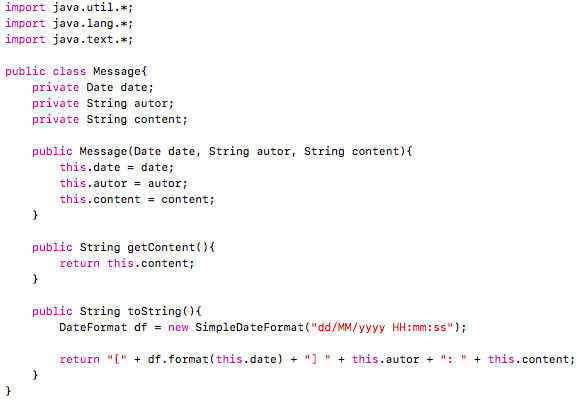


Figura 2 Representación de un mensaje

Una vez realizada la representación de un mensaje, se debe determinar quiénes son los que interactúan directamente con los mensajes, es decir, quiénes son los que participan dentro de la conversación. Para este programa, se separó en dos clases, un Usuario que desea comprar pasajes, y un chatbot, que hace de vendedor de pasajes. El método más importante de un usuario es *sendMessage()*, la cual permite a este intercambiar mensajes dentro del chat.

A diferencia del Usuario, la clase Chatbot posee una mayor lógica detrás, puesto que no es sólo enviar mensajes al azar, sino que se debe lograr una correcta interpretación de los mensajes del usuario, para de esta forma, permitir una conversación lógica y con sentido. Para esto, se ha realizado un método dentro de la clase Chatbot que permite intersectar strings, con el fin de identificar palabras “clave” que guien al Chatbot a responder de una manera más asertiva. Dado que la clase es de una extensión considerable, las siguientes figuras muestran los métodos más importantes para la correcta implementación del chat.

|  |
| --- |
| public List<String> intersect(String str){  List<List<String>> cities = Arrays.asList(Arrays.asList("Arica", "$32.200 pesos."), Arrays.asList("Iquique", "$30.100 pesos"), Arrays.asList("Antofagasta", "$21.600 pesos"), Arrays.asList("Copiapó", "$15.000 pesos"), Arrays.asList("La Serena", "$9.100 pesos"), Arrays.asList("Valparaíso", "$6.500 pesos"), Arrays.asList("Rancagua", "$3.000 pesos"), Arrays.asList("Talca", "$6.500 pesos"), Arrays.asList("Concepción", "$13.900 pesos"), Arrays.asList("Temuco", "$14.900 pesos"), Arrays.asList("Puerto Montt", "$19.900 pesos"), Arrays.asList("Coyhaique", "$33.000 pesos"), Arrays.asList("Punta Arenas", "$15.000 pesos"), Arrays.asList("Valdivia", "$17.900 pesos"));  List<List<String>> positive = Arrays.asList(Arrays.asList("sí"), Arrays.asList("si"));  List<String> splittedStr = Arrays.asList(str);  for (List<String> city : cities) {  if (str.toLowerCase().contains(city.get(0).toLowerCase())){  return city;  }  }  for (List<String> pos : positive){  if (str.toLowerCase().contains(pos.get(0).toLowerCase())) {  return pos;  }  }  if (str.toLowerCase().compareTo("no") == 0){  return Arrays.asList("no");  }  return null;  } |

Figura Método para intersectar mensajes del usuario

Figura Método que permite al Chatbot dar la bienvenida al Usuario.

La clase Log es casi una copia de algunos métodos de la clase *List*, puesto que la manera de llevar un registro de mensajes, ha sido a través de una estructura *List*, sin embargo, la clase Log se ha creado con la finalidad de mantener una mayor claridad y estructuración a la hora de implementar el código. La siguiente figura muestra el código fuente de la clase Log.

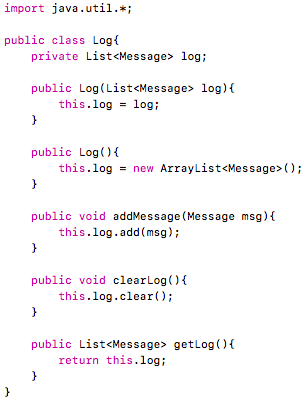


Figura Código fuente de la clase Log

Por un tema de orden y de buenas prácticas, la última clase corresponde a la clase *Main*, donde se instancia el laboratorio en general, y se ejecuta el chat, como muestra la Figura 6.

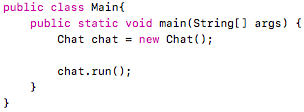


Figura Código fuente de la clase Main

Por último, las siguientes figuras muestran los algoritmos creados para implementar las instrucciones listadas en el capítulo 3.

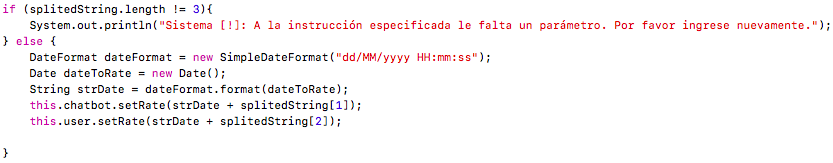


Figura Implementación instrucción !rate



Figura Implementación instrucción !saveLog

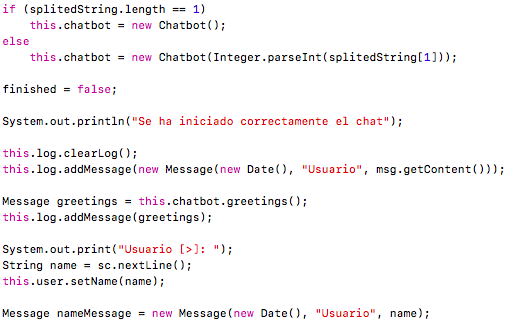


Figura Implementación instrucción !beginDialog seed

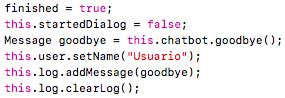


Figura Implementación !endDialog

# CAPÍTULO 5. RESULTADOS

A pesar de tener complejidades algorítmicas que podrían inducir a una lenta carga del chatbot, los tiempos de ejecución de las instrucciones no presentan un tiempo de respuesta notable para el usuario, siendo la creación del archivo log el que más tiempo demora.

La siguiente figura muestra el resultado presentado a la hora de utilizar la instrucción !beginDialog, en este ejemplo, sin el uso de una semilla.

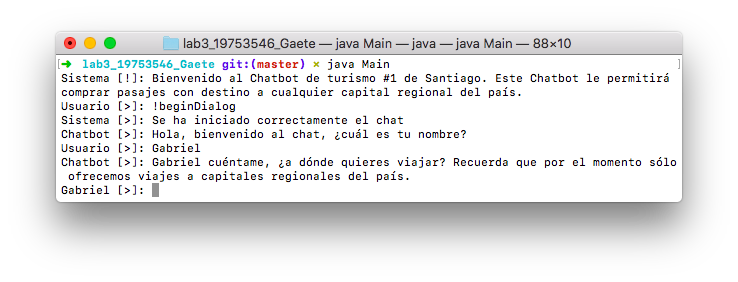


Figura Ejemplo de beginDialog

Por otra parte, a la hora de mantener una conversación entre usuario y chatbot, los tiempos de respuesta son relativamente rápidos. La siguiente imagen muestra el resultado de una conversación a través de su flujo esperado.

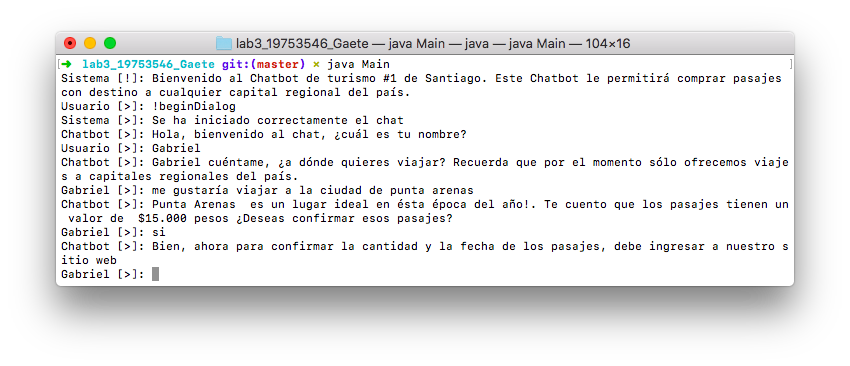


Figura Ejemplo de intercambio de mensajes

Además de permitir el intercambio de mensajes entre usuario y chatbot, manteniendo un flujo normal y coherente de una conversación, la instrucción ***!saveLog*** permite guardar un registro de lo conversado hasta el momento, como ejemplifican las siguientes figuras.

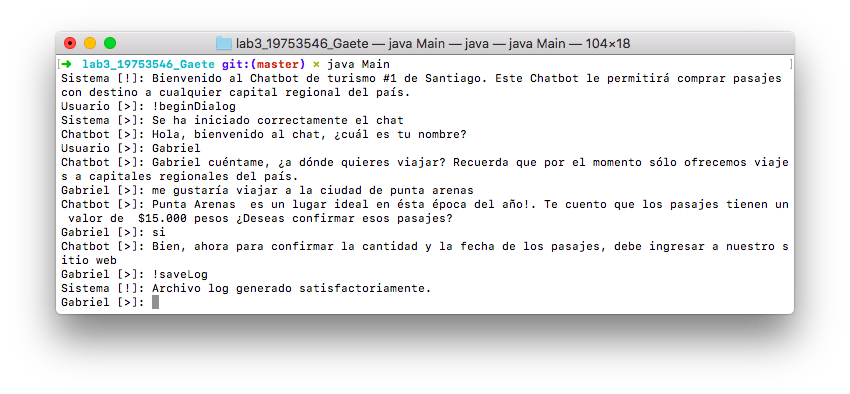


Figura Ejemplo de !saveLog

El archivo log generado en la figura 13, es registrado dentro de un archivo .log. Este archivo almacena la conversación que se lleva hasta el momento entre usuario y chatbot. Nótese que además de mantener el mensaje y el remitente, también se almacena cada uno de estos con su respectiva marca de tiempo, tanto para saber la fecha en que se tuvo una conversación en específico, como la hora en que el mensaje fue enviado. La siguiente figura permite observar el resultado generado dentro del archivo.

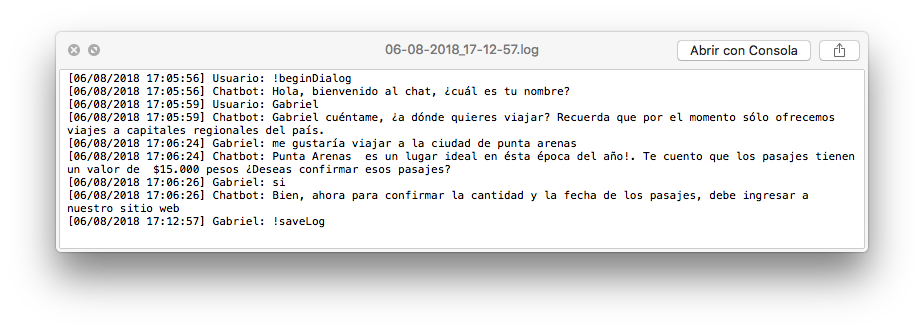


Figura Registro dentro del log

Por último, para terminar la conversación con el chatbot, es necesario utilizar la instrucción ***!endDialog***, sin embargo, el programa no se finaliza, pensando en que tal vez otro usuario (o el mismo si desea), pueda iniciar una nueva conversación con las instrucciones adecuadas.

Si se desea salir del programa, es decir, finalizar por completo la ejecución, se ha añadido una nueva instrucción, denominada ***!exit***, la cual permite esta acción. La siguiente figura muestra el resultado tanto de ***!endDialog*** como de ***!exit***.

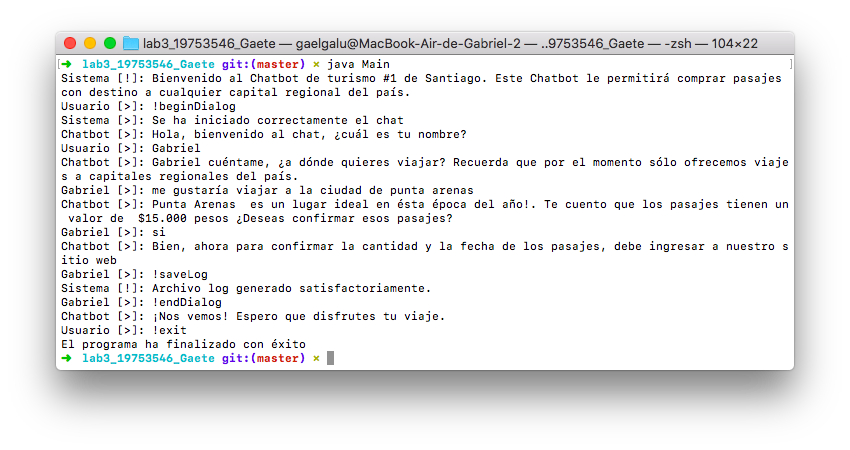


Figura Ejemplo de !endDialog y !exit

# CAPÍTULO 6. CONCLUSIONES

El paradigma orientado a objetos permite reflejar de una mejor manera la realidad, además de agrupar bajo un mismo elemento no sólo la información, sino también sus respectivos comportamientos y características (funciones propias, métodos, cambios de estado, incluso atributos), lo cual produce un mayor orden a la hora de estructurar soluciones, en contraste con los paradigmas vistos anteriormente, que no poseen ese nivel de estructuración.

Sin embargo, una desventaja evidente del paradigma orientado a objetos, es que la interpretación de un objeto queda totalmente a criterio del programador, por lo que dos programadores pueden llegar a ideas totalmente diferentes acerca de un mismo objeto, requiriendo una documentación más extensa en comparación a los paradigmas vistos anteriormente, como lo pueden ser el funcional y el lógico.

A pesar de la desventaja mencionada, el paradigma orientado a objetos sigue siendo una herramienta que permite un fácil mantenimiento, dado lo sencillo que resulta modificar las clases existentes.

Al ser comparado con los paradigmas vistos anteriormente (funcional y lógico), para la realización de este proyecto resulta una mejor herramienta el paradigma orientado a objetos, debido a la facilidad que nos entrega de abstraer el problema y llevarlo a una solución estructurada.

Finalmente, dado que se ha logrado desarrollar un *chatbot*  o *bot conversacional* como ha sido requerido, de paso demostrando los conocimientos expuestos en la cátedra de la clase y se ha obtenido el resultado esperado, es posible concluir que se ha logrado cumplir con el objetivo de este laboratorio. Sin embargo, no ha sido posible completar del todo los requerimientos pedidos (**loadLog**), debido a la forma en que la clase fue estructurada, se ha dificultado el proceso, siendo imposible concretar esta funcionalidad dentro de los plazos establecidos para el laboratorio.

# CAPÍTULO 7. REFERENCIAS

# Herrero, C. (s.f.). No son mis cookies. Recuperado el 21 de Abril de 2018, de No son mis cookies: http://nosinmiscookies.com/que-es-un-chatbot/