|  |  |
| --- | --- |
| **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: | 21 de Mayo del 2018 |

Santiago de Chile

1 - 2018

# TABLA DE CONTENIDO

[CAPÍTULO 1. INTRODUCCIÓN 4](#_Toc514554156)

[CAPÍTULO 2. MARCO TEÓRICO 5](#_Toc514554157)

[2.1 PARADIGMA LÓGICO 5](#_Toc514554158)

[2.2 PREDICADO 5](#_Toc514554159)

[2.3 CLÁUSULA DE HORN 5](#_Toc514554161)

[CAPÍTULO 3. DESCRIPCIÓN DEL PROBLEMA 6](#_Toc514554162)

[3.1 ANÁLISIS 7](#_Toc514554163)

[CAPÍTULO 4. DESCRIPCIÓN DE LA SOLUCIÓN 8](#_Toc514554164)

[CAPÍTULO 5. RESULTADOS 11](#_Toc514554165)

[CAPÍTULO 6. CONCLUSIONES 14](#_Toc514554166)

[CAPÍTULO 7. REFERENCIAS 15](#_Toc514554167)

# TABLA DE FIGURAS

[Figura 1 Representación de un mensaje 8](#_Toc514638895)

[Figura 2 Código fuente de beginDialog 8](#_Toc514638896)

[Figura 3 Código fuente de endDialog 9](#_Toc514638897)

[Figura 4 Código fuente de sendMessage 9](#_Toc514638898)

[Figura 5 Código fuente de logToStr 9](#_Toc514638899)

[Figura 6 Código fuente de test 10](#_Toc514638900)

[Figura 7 Ejemplo de beginDialog 11](#_Toc514638901)

[Figura 8 Ejemplo de sendMessage 12](#_Toc514638902)

[Figura 9 Ejemplo de endDialog 12](#_Toc514638903)

[Figura 10 Ejemplo de logToStr 13](#_Toc514638904)

[Figura 11 Ejemplo de test 13](#_Toc514638905)

# 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 *“Paradimas 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 *log* o historial de conversaciones, y un usuario. 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). De esta forma, un diagrama inicial de clases sería como muestra la siguiente figura:

**INSERTAR DIAGRAMA DE CLASES INICIAL**

# 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 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, sólo quiere intercambiar un mensaje con el bot.

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

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*.

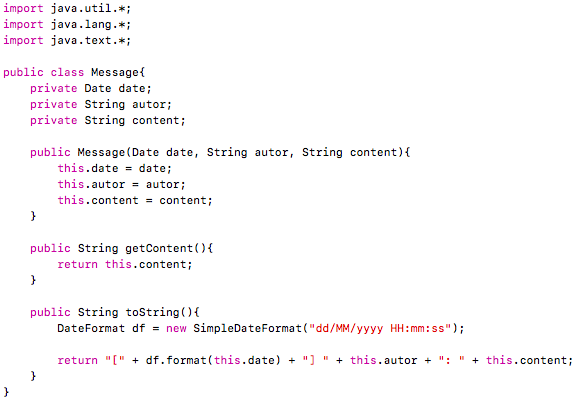


Figura 1 Representación de un mensaje

De esta forma, al momento de crear una representación para el log, se puede acceder a toda la información de los mensajes a través de los selectores para este TDA, logrando que los predicados que deben retornar un log actualizado (**beginDialog, sendMessage, endDialog**) tengan una manera directa de trabajar con este. Las siguientes figuras muestran el código fuente de **beginDialog**  y de **endDialog**, cuya lógica y funcionamiento es similar.

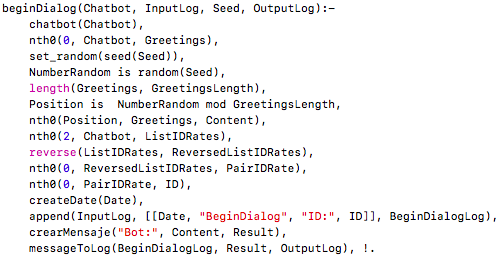


Figura 2 Código fuente de beginDialog

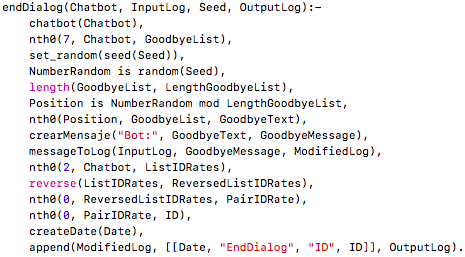


Figura 3 Código fuente de endDialog

Por otro lado, **sendMessage** requiere pasos adicionales. Para lograr interpretar los mensajes que entrega el usuario, con el fin de entregar una respuesta que se acomode al flujo de la conversación, se trata al contenido del mensaje del usuario como una lista de strings; luego estos son intersectados con una series de palabras para interpretar el mensaje. Una vez interpretado el mensaje, se determina la respuesta que debe entregar el chatbot. La siguiente figura muestra el código fuente de este predicado.

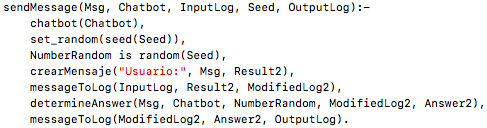


Figura 4 Código fuente de sendMessage

Una vez realizados los predicados descritos anteriormente, la implementación de **logToStr** se reduce a iterar recursivamente sobre el log, concatenando directamente cada TDA Mensaje presente en su interior a un string. Para eso, se utiliza un predicado auxiliar que permita la concatenación, como lo muestra la siguiente figura.



Figura 5 Código fuente de logToStr

Teniendo todas estas funcionalidades, el predicado **test** ya puede ser implementado. La siguiente figura, muestra el código fuente de dicho predicado.

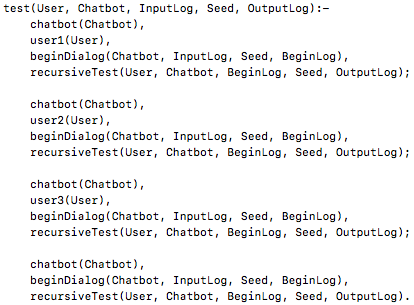


Figura 6 Código fuente de test

Nótese que en la figura 6, se tienen distintas definiciones de usuario. Esto se da porque por enunciado, se pide entregar al menos 3 ejemplos de usuarios distintos. Sin embargo, se agrega la posibilidad de que el usuario ingrese una lista de strings “personalizada”, es decir, que no esté dentro de la base de conocimientos del programa.

# CAPÍTULO 5. RESULTADOS

Debido a la naturaleza de lo que es un chatbot, este no responde de manera booleana (Verdadero o Falso), sino que debe ser capaz de generar texto con el cual se permita la conversación con el usuario. Se puede apreciar en la siguiente figura, los resultados que entrega la utilización del predicado **beginDialog**. Cabe destacar, que como el usuario no conoce la representación interna de la estructura Chatbot, esta no es necesaria de ingresar, por lo que *prolog* también mostrará, aparte el *log* de salida, la correspondiente estructura chatbot utilizada para llevar a cabo la conversación.

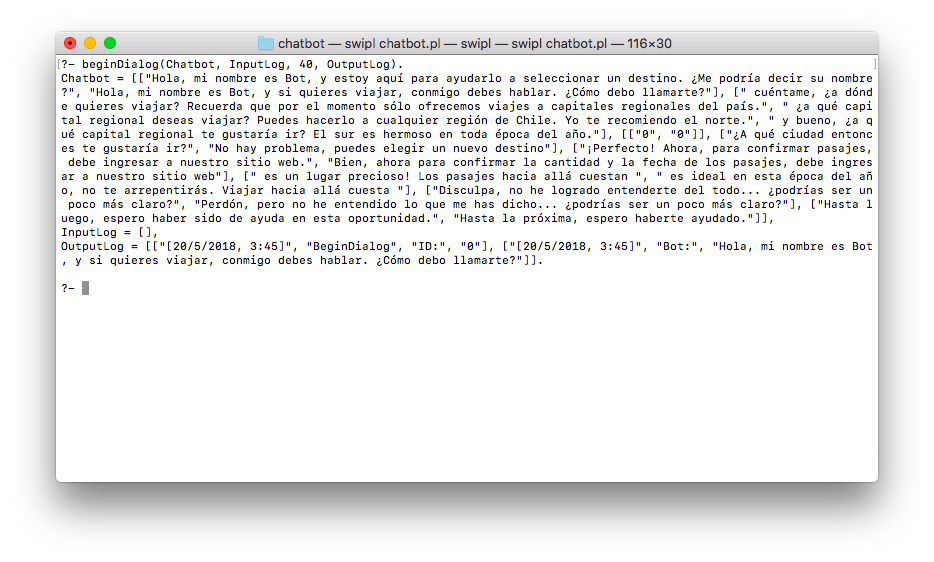


Figura 7 Ejemplo de beginDialog

Por otra parte, para **sendMessage**, se necesita el resultado del OutputLog del predicado anterior. En la siguiente figura, se ha copiado el outputLog de la Figura 7, produciendo el siguiente resultado.

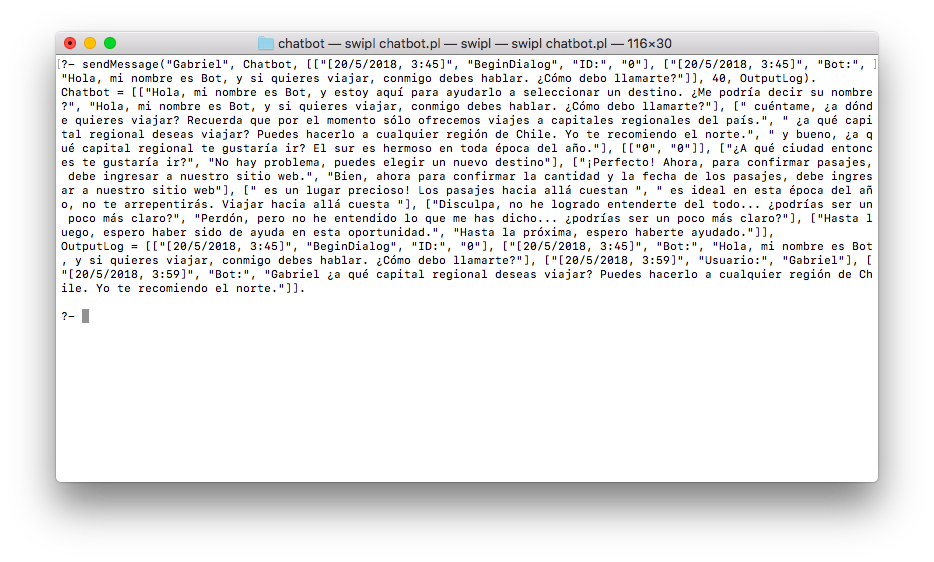


Figura 8 Ejemplo de sendMessage

La siguiente imagen muestra el resultado del predicado **endDialog.** Nuevamente, el resultado importante para mantener el flujo de la conversación, es ***OutputLog.***

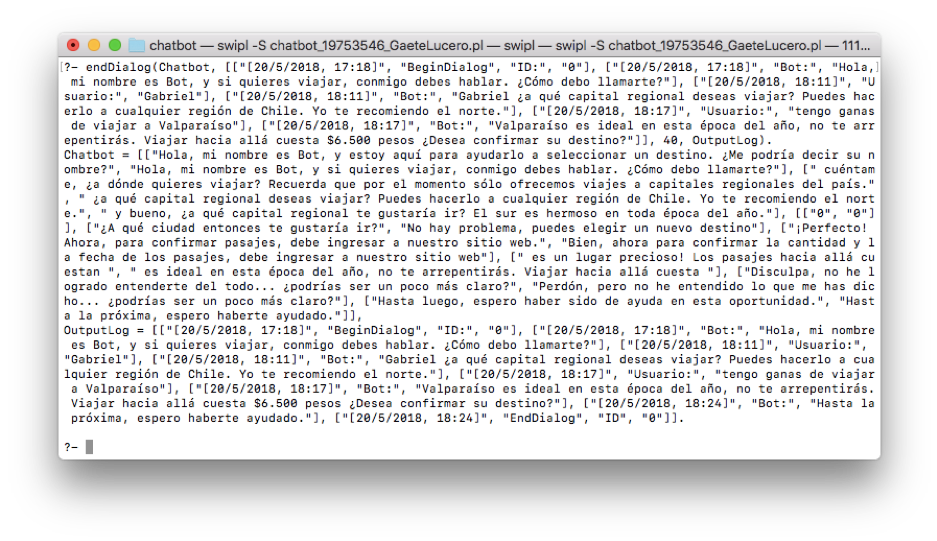


Figura 9 Ejemplo de endDialog

La siguiente figura, muestra el resultado generado por la funcionalidad **logToStr**, dado cierto log de entrada.

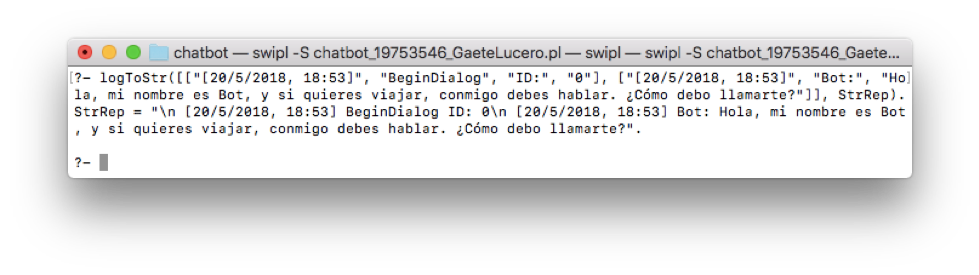


Figura 10 Ejemplo de logToStr

Por último, el predicado **test** se muestra en la siguiente figura. Nótese que, en caso de no especificarse entrada de User, permite simular las tres conversaciones previamente definidas por el programador.

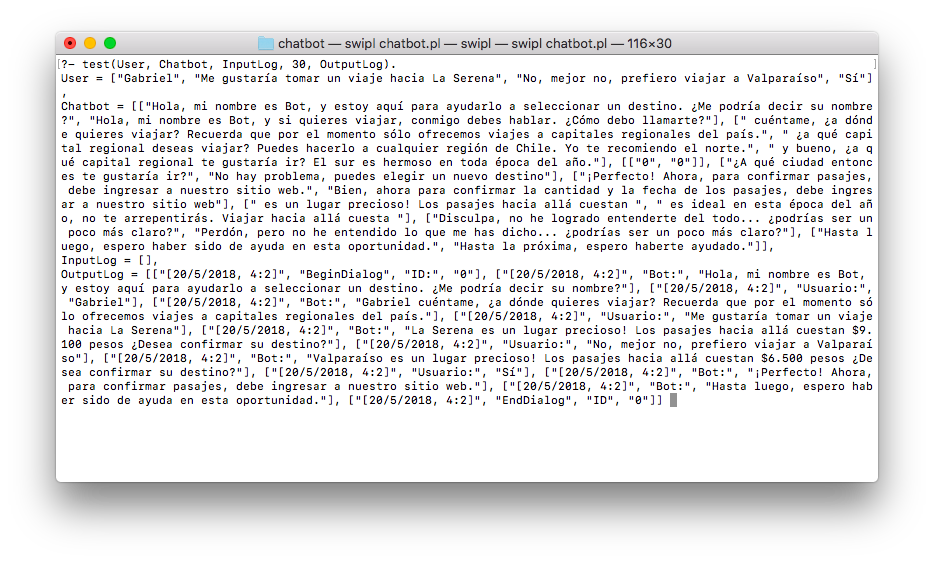


Figura 11 Ejemplo de test

# 

# CAPÍTULO 6. CONCLUSIONES

El paradigma lógico tiene la ventaja de ser simple, los “algoritmos” pueden mejorarse al modificar las componentes de control sin necesariamente cambiar la lógica del programa. Este paradigma presenta además una sencillez para representar estructuras de datos complejas, sin tener la necesidad definir estas estructuras, como se podría hacer en otros paradigmas, por lo que se presenta una increíble potencia para resolver problemas a gran escala, sólo con definir relaciones entre los elementos relevantes. Sin embargo, por lo mismo, puede pasar a ser una mala opción para la resolución de algunos problemas, puesto que se deben definir ABSOLUTAMENTE TODAS las relaciones en la base de conocimiento para poder establecer las reglas y trabajar en base a éstas, teniendo en cuenta que si esta base de conocimiento no posee la información suficiente, las respuestas pueden retornar valores no esperados.

Dado que un chatbot se basa en el reconocimiento de lenguaje natural, permitiendo procesar respuestas frente a mensajes de un usuario, es que en particular *Prolog* es una buena herramienta, ya que en la base de conocimientos se pueden definir las palabras que el chatbot puede llegar a reconocer, siendo este un buen lenguaje para el procesamiento de lenguaje natural.

Al ser comparado con el paradigma funcional, para este laboratorio, resulta ser una mejor herramienta el paradigma lógico, por lo expresado anteriormente.

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.

# 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/