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

Informe Laboratorio 3
Paradigma Orientado a Objetos
Sistema de creación, despliegue y administración de chatbots

Stephan Paul Ramirez

Fecha:

10 de diciembre 2023

UdeSantiago de Chile

Índice de Contenidos

1. Introducción....2
2. Descripción del problema....2
3. Descripción del paradigma....2
4. Análisis del problema....3
5. Diseño de la solución....4
6. Aspectos de implementación....5
7. Instrucciones de uso....5
8. Resultados y autoevaluación....6
9. Conclusión....6
10. Bibliografía....7
11. Anexos....7

UdeSantiago de Chile

1. INTRODUCCIÓN

En el presente informe se documenta el proceso de realización del tercer laboratorio de la asignatura paradigmas de programación. El objetivo del laboratorio consiste en implementar un ambiente contenedor de chatbots en java, usando el paradigma orientado a objetos, que permita a un usuario crear e interactuar con distintos chatbots, en donde cada uno contendrá a su vez distintas preguntas de distintos temas y distintas opciones asociadas a otros chatbots, así de esta forma se pueden enlazar las opciones con otras preguntas y así simular una conversación con un usuario. Las operaciones que se abarcan con respecto a crear un chatbot son; identificar un chatbot, añadir preguntas al mismo, añadir opciones a las preguntas y especificar enlaces para el flujo de interacción dentro del chatbot hacia otros chatbots. Luego con respecto a las interacciones del chatbot se tiene que mediante una serie de opciones el usuario pueda escoger alguna de estas para seguir la conversación. También se considera como operación la posibilidad de mostrar una síntesis de la conversación o historial de algún usuario.

2. Descripción del problema

A partir de lo anteriormente mencionado, es que el problema que se plantea en el laboratorio consiste en cómo se podría implementar una idea de solución para crear el sistema contenedor de chatbots usando el paradigma orientado a objetos en el lenguaje de java, luego esto llevaría plantearse una serie de preguntas: ¿Cómo se podría "traducir" las ideas de solución diseñada en el primer y segundo laboratorio a este lenguaje? ¿Cómo se podría implementar el uso de objetos y clases para representar los chatbots?; sin embargo, debido a que se cuenta con los ejemplos del menú y los requerimientos de el enunciado, es que se puede contestar que se deben implementar clases siguiendo la lógica de los anteriores laboratorios en java adaptándose a los nuevos requerimientos usando las herramientas que ofrece el lenguaje, ya sea que en vez de usar listas para almacenar un conjunto de datos, ahora se usen los atributos de un objeto, o eliminar duplicados con un ciclo for que revise condiciones de un objeto con un método selector de algún atributo, etc. Junto con que posteriormente se debe mostrar el procedimiento de estos requerimientos de manera interactiva usando un menú que imprima por consola los resultados de alguna operación.

3. Descripción del paradigma

El paradigma orientado a objetos es el paradigma con el cual se aborda el proyecto, este consiste en describir la construcción de un programa basado en "objetos", es decir, se basa en el pensamiento de que todo es un objeto, al igual que todo lo que nos

rodea en el mundo real también lo es. Estos objetos a su vez poseen “atributos”, que básicamente son el diferenciador que estos objetos pueden tener hacia otros, como lo pueden ser; su color, su altura o espesor. Por último, hay que mencionar que estos objetos también poseen “comportamientos”, que definen las acciones que un objeto puede realizar (Esteban, 2020).

En el contexto del proyecto este paradigma se emplea usando java en el IDE intellij, el cual está fuertemente basado en el paradigma orientado a objetos, un programa en java esta fuertemente asociado a lo que es un diagrama de clases. Un diagrama de clases en UML es un tipo de diagrama de estructura estática que describe la representación de un sistema basado en la programación orientada a objetos. El diagrama de clases representa el sistema describiendo las partes que la componen, como lo son las clases, atributos, operaciones y las relaciones entre objetos (Esteban, 2020).

Luego como el problema implica almacenar cosas o “agregarlas”, se usará el concepto de agregación en clases, en dónde se usará la clase List, para almacenar los distintos objetos como atributo de otro, con el fin de que un sistema tenga como atributo una lista de chatbot, luego que este último tenga como atributo una lista de flujos y así.

4. ANÁLISIS DEL PROBLEMA

Identificando los principales componentes de lo que debería ser la implementación de la solución se puede determinar que usando como estructura base las clases, se deberá implementar un sistema “global” que sea una clase que tenga como atributo una lista que contenga todos los chatbots del sistema, en este contexto se deberán usar métodos que construyan estas estructuras asignando las entradas a cada atributo, y teniendo en cuenta que dentro de los requerimientos se especifica que cada estructura deberá tener un id único para su conjunto, eso quiere decir que dentro de un flujo por ejemplo, todas las opciones deberán tener un id único, esto implicaría en java que en el método constructor de un objeto, se deba verificar si la lista de opciones que se tiene como entrada tengan identificadores distintos, recorriendo la lista y hacer una nueva sin repetidos, que verifique con un método si es que ya está repetido.

Empezando por el primer RF, como se tienen distintas estructuras con distintas operaciones, se deberá implementar una manera de plasmar en el código cada una de estas por separado, en este sentido se deben especificar distintos tipos de datos abstractos en donde cada uno tendrá su representación, constructor, selectores, modificadores y otros métodos asociados. Luego como se mencionó anteriormente, se deberán implementar métodos que sigan estas operaciones de cada TDA siguiendo las reglas de duplicados, es decir, que los constructores mantengan la primera

instancia si es que existen, lo mismo para los modificadores del estilo “add”, junto con que también como se debe interactuar con un usuario, se debe implementar una manera de que un usuario usando el menú pueda registrarse en el sistema, iniciar sesión, cerrar sesión y que contenga su historial en el mismo, y que luego estos menú se muestren diferenciando un usuario común a uno administrador (ver Imagen 1).

5. DISEÑO DE LA SOLUCIÓN

Empezando por las clases y estructuras, simplemente se optó por crear interfaces que especifican las operaciones que utilizara cada clase. Para el método constructor de option se optó por hacer un método que recibe como parámetro los atributos del option y luego asigna el atributo del option a la entrada usando “this.atributo” y una asignación a la entrada correspondiente. Para el constructor de Flow es lo mismo, pero se revisa que la lista de options de la entrada no tenga repetido creando una lista vacía en donde se agregan las opciones de entrada con un ciclo, en donde se revisa primero si es que la lista creada no contenga ya el option actual usando “contains”, que a su vez usa el “equals” de option, el cual revisa si el id es el mismo o no. Después para agregar una opción a un flujo, se opta por usar la misma lógica del constructor con “contains” pero solo verificando unicidad con el option de entrada. De manera similar a Flow se trabaja con chatbot y chatbotAddFlow pero verificando flows. Para los usuarios se optó por hacer una interfaz usuario y que luego esta se implemente en 2 clases distintas, usuario común y administrador, en donde cada uno tiene como atributo su username y su chatHistory asociado, que es un string formateado para usar “synthesis” y mostrarlo por consola. Para system, se trabaja de manera similar para verificar unicidad de chatbots, pero ya que la implementación de este puede tener más libertad, se optó por agregar 2 atributos más aparte de los argumentos recibidos, siendo estos la lista con los usuarios registrados y la lista con actual (lista con los id del chatbot y flujo actual de talk) y por otro lado el otro atributo agregado es un atributo que indica el usuario loggeado guardando su username. (ver imagen 1)

Continuando con “systemAddUser” y “systemAddChatbot”, se hizo un método similar a los otros “add” usando “contains”, pero User se revisa si el username es el distinto para agregarlo. Luego para “systemLogin” y “systemLogout”, se optó por reemplazar el atributo de la sesión actual con el nombre del usuario ingresado para iniciar y para cerrar se asigna un String vacío. Para el método talk, se agrega otro método a system, en donde la implementación consiste en verificar que el mensaje del usuario haga match con algún option del flujo actual y luego vincular el nuevo flujo reemplazando actual por una lista del “chatbotcodelink” e “initialflowcodelink” de la opción con la que hizo match, por último, “synthesis”, busca el username de la entrada en la lista de registrados y retorna su chatHistory. Luego para el menú se usó un do while en donde la condición de salida es el numero de la opción salir, y anidado

a un switch que es para proceder dependiendo de la opción del usuario, en donde se puede anidar a otros menús que imprimen indicaciones en la pantalla con “println”, los cuales son métodos miembros de la clase menú, y para marcar si una operación se realizó o no, algunos métodos de los RF retornan booleanos (ver Imagen 2).

6. ASPECTOS DE LA IMPLEMENTACIÓN

Con respecto a la estructura del proyecto, primero se abordaron los requerimientos funcionales luego de diagramar el UML de análisis, empezando por las clases y estructuras, se crearon interfaces para las operaciones de cada clase y luego se empezó a trabajar los métodos de manera ordenada, es decir de 1 en 1 cada día y realizando los commit correspondientes por cada avance en GitHub, junto con que se separaron las entidades en archivos con la extensión java los cuales implementan las interfaces mencionadas. Después de terminar la parte funcional, se trabajó en el menú, de manera que al final se organizó una carpeta llamada “codigofuente” que contiene las interfaces, las clases que los implementan y “main” (cabe destacar que cada archivo también tiene el formato de nombre “nombre_rut_apellidos.extension”). Luego se realizó la autoevaluación y el UML de diseño, para finalmente hacer el informe del proyecto. Con respecto a el IDE, lenguaje y versión utilizados, se usó IntelliJ IDEA Community 2023.3 para programar en java, utilizando el JDK azul-11.0.21. Cabe recalcar que todo esto se desarrolló en el OS de Windows 11.

7. INSTRUCCIONES DE USO

Para el uso adecuado del sistema, se debe extraer el archivo zip y luego usar el comando cd en el cmd con la dirección en donde se extrajo el zip del proyecto (cd .../Dirección), para luego usar los comandos con los cuales se revisará el proyecto, en Windows, es decir, “gradlew.bat build” para compilar y “gradlew.bat run” para ejecutar, luego en el archivo “léeme” están las instrucciones completas del programa como tal, sin embargo, solo hay que seguir las instrucciones que se imprimen por consola, ingresando los números asociados a la opción que se quiere elegir con “enter” en el cmd, hasta que se decida ingresar un nombre de usuario o hablar con el chatbot en donde se debe ingresar texto. Con respecto a posibles errores, se recomienda seguir las instrucciones y no ingresar caracteres especiales para evitarlos.

UdeSantiago de Chile

8. RESULTADOS Y AUTOEVALUACIÓN

Clases y estructuras	Menú interactivo	option	flow	flowAddOption	chatbot
1	0.75	1	1	1	1
chatbotAddFlow	usuario	system	systemAddChatbot	systemAddUser	systemLogin
1	1	1	1	1	1
systemLogout	talk	synthesis	simulate		
1	1	1	0		

Con respecto al 0.75 del Menú, esto es debido a que la única forma de dejar de hablar con el chatbot es ingresando una keyword especial, "SALIR" y por otro lado es porque a pesar de que el menú indica la opción para inicializar el sistema o crearlo, este no deja crear uno totalmente nuevo debido al diseño de la solución. No se completó "simulate" porque no se entendió del todo como implementar este requerimiento.

9. CONCLUSIÓN

Para finalizar se puede decir que el resultado obtenido cumple con casi todos los requerimientos fundamentales de lo requerido, exceptuando la simulación, ya que el sistema permite crear chatbots con el menú, permite añadir preguntas, permite añadir opciones, los enlaces quedan en los atributos de una opción y se puede interactuar usando las palabras clave y el número de la opción. Algunas de las limitaciones del programa son el hecho de que para dejar de interactuar con un chatbot requiere ingresar la keyword indicada "SALIR". Comparando con el paradigma lógico y el funcional se tiene que una ventaja con respecto a estos es que, al ser imperativo, se puede revisar el conjunto de listas con ciclos y de esta manera se puede agilizar la verificación de duplicados, junto con que los getters y setters se podían hacer de manera sencilla usando el concepto de los atributos. Luego otra ventaja con respecto a los otros paradigmas es que, a diferencia de estos, en el orientado a objetos se pueden concatenar operaciones se puede hacer de manera tan conveniente como llamar método tras método a un objeto en el cual se quiere operar, junto con que usar métodos cambia de manera directa un atributo de un objeto si se usa "this" en este. Un beneficio del paradigma fue que como era requerimiento de esta entrega el hecho de mostrar cosas por consola se podía saber de manera rápida como respondía el sistema al operar con el y si fue exitosa la operación o no.

10. BIBLIOGRAFÍA

- Esteban, D. (2023) *Fundamentos de la Programación Orientada a objetos (poo)*, *Medium*. Available at: https://medium.com/@diego.coder/fundamentos-de-la-programaci%C3%B3n-orientada-a-objetos-poo-5f8585346e92 (Accessed: 10 December 2023).

11. ANEXOS

- Imagen 1

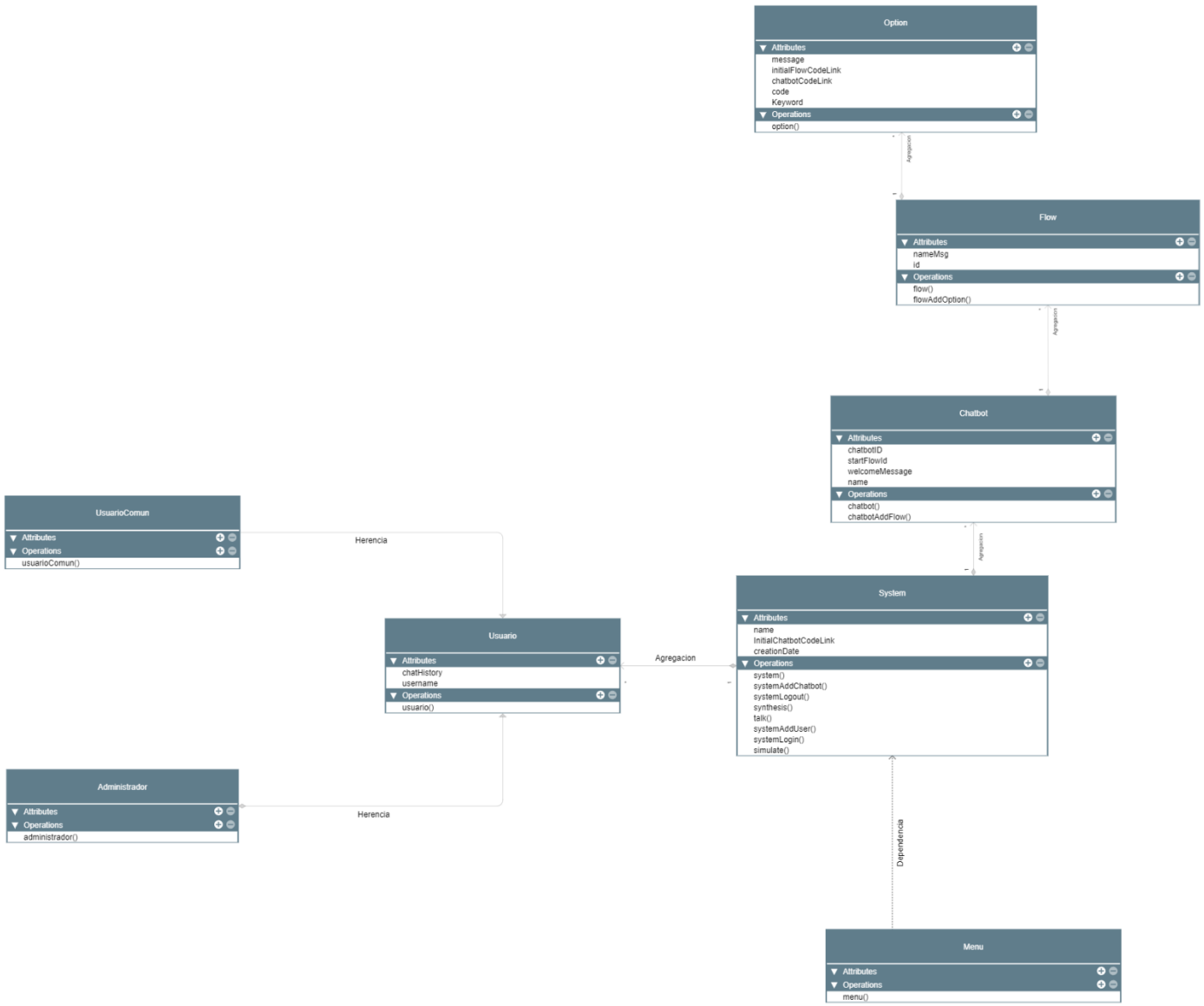

Diagrama de Análisis

UdeSantiago de Chile

- Imagen 2

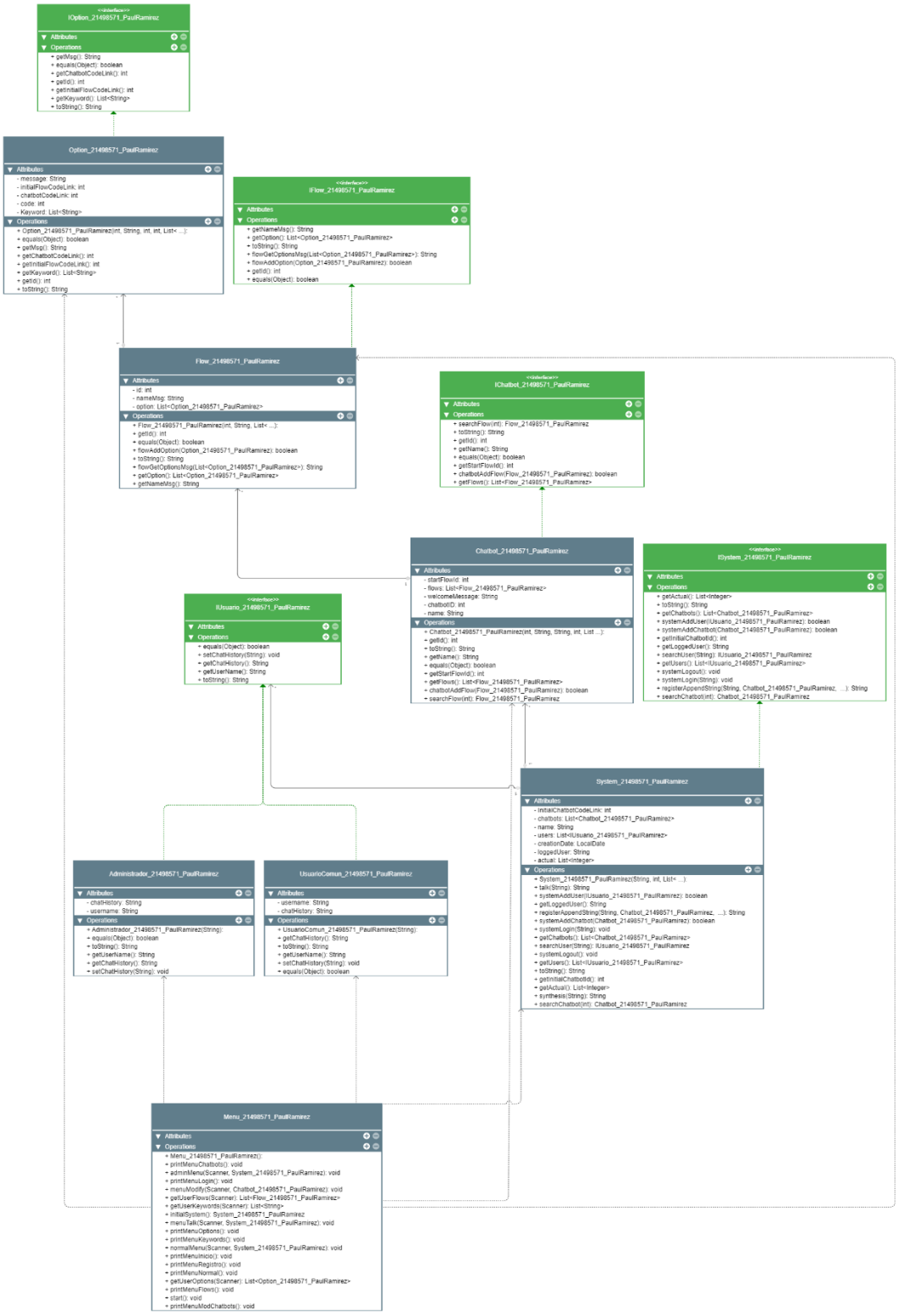

Diagrama de Diseño