DEPARTAMENTO DE
INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

Paradigmas de programación Laboratorio 3 (Paradigma Orientado a Objetos – Lenguaje Java)

Estudiante: Elías Zúñiga Tobar

Sección: B-2

Profesor: Gonzalo Martínez

Fecha: 11 de diciembre de 2023

Índice

Introducción

Este informe tiene como objetivo el de aprender, entender y explicar sobre el uso del paradigma orientada a objetos a través del lenguaje de programación “Java” usando el programa de “Intellij IDEA” para el desarrollo de este laboratorio, en donde se tomarán conceptos que no se aplican en otros paradigmas. Se dará a conocer la problemática de este trabajo junto con su análisis, el paradigma a que se usará, dar a conocer los conceptos que abarca, explicar la solución que se implementará y que después se evaluará y por último se determinará las conclusiones correspondientes.

Problema: En este trabajo se pide el desarrollo e implementación de un sistema TDA para la creación, despliegue y administración de chatbots simplificado, donde un usuario realizar operaciones tales como crear chatbots, interactuar con ellos y ofrecer una síntesis de las interacciones con el bot. Para ello, se necesitará los siguientes elementos:

- **Sistema (System):** Elemento indispensable para la implementación de la solución, se usará como base en donde se agregarán y guardarán los chatbots, los usuarios registrados en el sistema junto con su historial de chat y también su fecha de creación.
- **Chatbot:** Corresponde a los chatbots que se agregarán al sistema, los cuales tendrán una ID, un mensaje entrante y los flujos.
- **Flujo (Flow):** Los flujos serán aquellos elementos que guardarán las opciones, estos contarán también de una ID, un mensaje entrante.
- **Opciones (Option):** Estos serán las opciones que el usuario tendrá que escoger según el chatbot con el que está interactuando, tendrán una ID, un mensaje, una pertenencia a un flujo, chatbot y palabras claves.
- **Usuario (User):** Serán los usuarios que serán registrados en el sistema, tendrán una ID y estos pueden ser usuarios normales o administradores.

Sin embargo, este problema implicará algunas limitaciones debido al uso del paradigma y lenguaje de programación que se explicará a continuación.

Paradigma: En este trabajo se dará el uso del “Paradigma Orientada a Objetos” o POO, el cual se centra en la manipulación de objetos los cuales son entidades que combinan datos y comportamientos en un solo paquete. La POO organiza el código de manera modular al combinar datos y comportamientos en objetos, lo que facilita la creación, mantenimiento y reutilización de código en el desarrollo de software. Algunos lenguajes aparte de Java que aplican POO serían Python, C++, C#, entre otros.

- **Clases:** Define de manera genérica cómo van a ser los objetos de un determinado tipo. Por ejemplo, una clase para representar a animales puede llamarse ‘animal’.

- **Atributos:** Las clases pueden tener atributos, también conocidos como variables de instancia, que representan las características o propiedades del objeto.
- **Métodos:** Los métodos son funciones que pueden realizar acciones y manipular los atributos de la clase.
- **Relaciones:** En el contexto de POO, todas las clases están vinculadas entre sí mediante relaciones adecuadas. Estos enlaces ayudan al usuario a comprender a fondo la conexión entre diferentes entidades. Estos pueden ser:
    - Asociación.
    - Agregación.
    - Composición.
    - Herencia.
    - Dependencia.
    - Interface.

Desarrollo

Análisis del problema:

Se solicitó la implementación de este sistema de chatbots a través del software de programación “IntelliJ”, y esta implementación debe cumplir que se pueda crear chatbots, administrarlos e interactuar con estos. Para ello el elemento “Sistema” se usará como base para la creación y agregado de chatbots, flujos, opciones y usuarios. Sin embargo, para la interacción con algún chatbot se necesita que un usuario haya iniciado sesión en el sistema.

Los métodos requeridos (Anexo 1) para la implementación son las siguientes:

1. **option:** Método constructor de una opción para flujo de un chatbot. Cada opción se enlaza a un chatbot y flujos especificados por sus respectivos códigos.
2. **flow:** Método constructora de un flujo de un chatbot.
3. **flowAddOption:** Método modificador para añadir opciones a un flujo.
4. **chatbot:** Método constructor de un chatbot.
5. **chatbotAddFlow:** Método modificador para añadir flujos a un chatbot.
6. **system:** Método constructor de un sistema de chatbots. Deja registro de la fecha de creación por.
7. **systemAddChatbot:** Método modificador para añadir chatbots a un sistema.
8. **systemAddUser:** Método modificador para añadir usuarios a un sistema.
9. **systemLogin:** Método que permite iniciar una sesión en el sistema.
10. **systemLogout:** Método que permite cerrar una sesión abierta.
11. **systemTalk:** Método que permite interactuar con un chatbot.

12. **systemSynthesis:** Método que ofrece una síntesis del chatbot para un usuario particular partir de chatHistory contenido dentro del sistema.
13. **systemSimulate:** Permite simular un diálogo entre dos chatbots del sistema.

Con esto dicho se realizó un diagrama de análisis (Anexo 2).

Diseño de solución:

Con lo que se solicitó se implementó los siguientes TDA utilizando los elementos anteriormente mencionados:

- **TDA_system:** Clase de objeto que contiene como atributos los usuarios registrados, el usuario que inició sesión y una lista de objetos de clase Chatbot, una lista de Strings que será el historial de interacción entre el usuario logeado y un chatbot, su fecha de creación y un arreglo de 2 elementos que será la posición para guiar hacia dónde va una interacción entre un chatbot y un usuario.
- **TDA_chatbot:** Clase de objeto que contiene como atributos el nombre del chatbot en sí y una lista de flujos.
- **TDA_flow:** Clase de objeto que contiene como atributos una ID, un mensaje entrante y una lista de objetos de clase Option.
- **TDA_user:** Clase de objeto que contiene como atributos una ID, el nombre del usuario y un booleano que verifica si el usuario es normal o administrador.
- **TDA_option:** Clase de objeto que contiene como atributos una ID, un mensaje y palabras clave.

La implementación consiste en la creación de objetos donde cada uno pertenece a una clase en cada archivo TDA, es decir, un objeto de clase Option, Flow o Chatbot, entre otros. Con este paradigma permite un mejor manejo de variables al momento de codificar al estar relacionado a la programación imperativa.

Aspectos de implementación:

Para este laboratorio se usó el software “Intellij IDEA 2023.2.4” donde se trabajó el lenguaje Java utilizando la versión 11 del JDK. Cada archivo “Class” de los TDA junto con su archivo “Interface” (a excepción del archivo Main.java) se encuentran funciones que trabajan con los datos correspondientes a su campo (Anexo 3). Además de implemento un menú donde permite registrar usuarios como usuarios normales o administradores, donde un usuario normal solo se limitaría a interactuar un chatbot y consultar una síntesis según su historial. Mientras tanto, el administrador tiene la capacidad de crear o modificar opciones, flujos y chatbots, además de interactuar con los chatbots, y todo esto dentro de un sistema único de chatbots. Sin embargo, no cuenta con una simulación de interacción entre 2 chatbots.

Instrucciones de uso:

Al ejecutar el programa se le entregará una interfaz con las opciones de registrar, iniciar sesión y salir, para este tipo de opciones se le recomienda solo escribir números ya que si se le escribe un string el programa dará error. Y ya al momento de que se esté interactuando con un chatbot, ya se permite escribir tanto strings como números.

Resultados y autoevaluación:

Al ejecutar el programa se probó con registrar 2 usuarios siendo “usuario1” un administrador y “usuario2” uno normal, con el usuario administrador de logro crear y modificar chatbots al igual que los flujos. mediante el inicio de sesión del usuario2 se logró probar el interactuar con los chatbots y registrar el historial de interacción. Por lo tanto, los métodos implementados (desde option hasta systemSynthesis) dentro del plazo establecido entregan correctamente el sistema esperado, pero nunca se logró implementar una simulación de interacción entre 2 chatbots por falta de ideas.

Conclusión

Con lo obtenido anteriormente, se puede concluir que se ha logrado cumplir la mayor parte de lo solicitado para este laboratorio, el programa implementado retorna un sistema de chatbots de forma correcta y se puede interactuar con ella, pero no hay forma de simular una interacción entre 2 chatbots.

Durante el desarrollo del código fuente se han enfrentado dificultades, tales como al consultar una variable mediante un selector, este terminaba en un ciclo de recursiones infinitas o cuando el programa daba error cuando comparaba variables de diferente tipo, por ejemplo, comparar un Integer con un String.

Además, el uso del paradigma funcional a veces limitaba el cómo implementar algunas funciones generando en algunos casos una parálisis paradigmática.

Bibliografía

- Miriam Martínez (2020, noviembre 2) *¿Qué es la programación orientada a objetos?* Recuperado de https://profile.es/blog/que-es-la-programacion-orientada-a-objetos/

- n. a. (n. d.) *Diagramas de clase UML* Recuperado de https://www.edrawsoft.com/es/article/class-diagramrelationships.html#:~:text=Es%20una%20relación%20que%20vincula,discontinua%20con%20una%20flecha%20hueca.

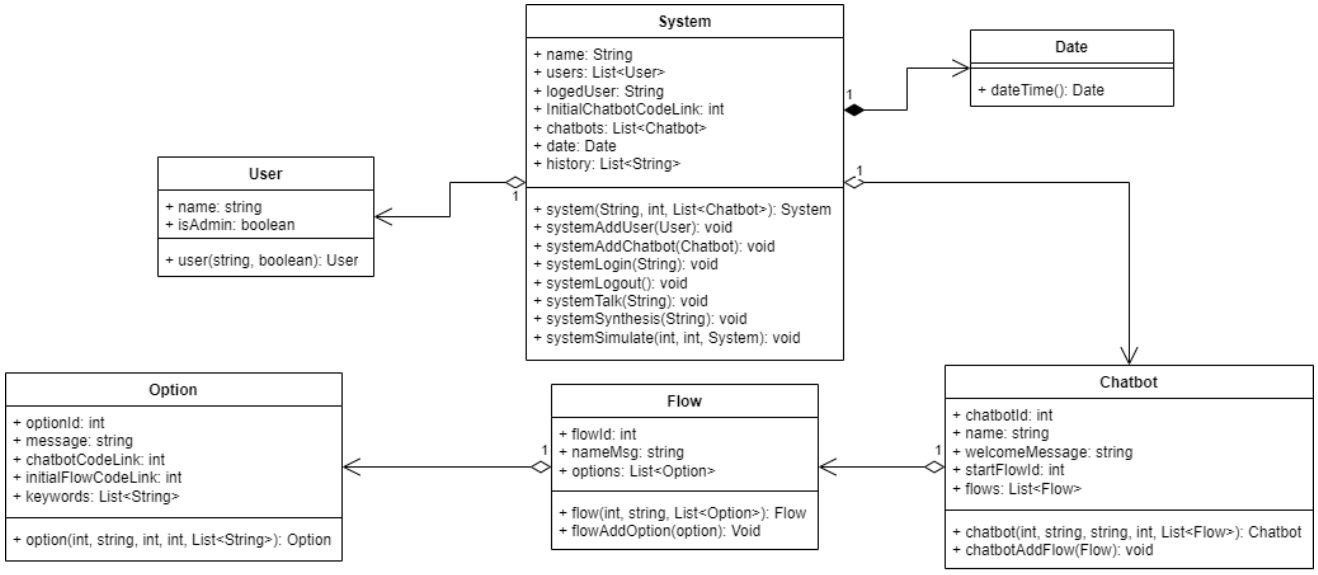
- Jorge López (2023, Octubre 27) *Introducción a POO en Java: Objetos y clases* Recuperado de https://openwebinars.net/blog/introduccion-a-poo-en-java-objetos-y-clases/#:~:text=La%20Programación%20Orientada%20a%20Objetos%20(POO)%20es%20un%20enfoque%20fundamental,una%20instancia%20de%20una%20clase.

Anexos

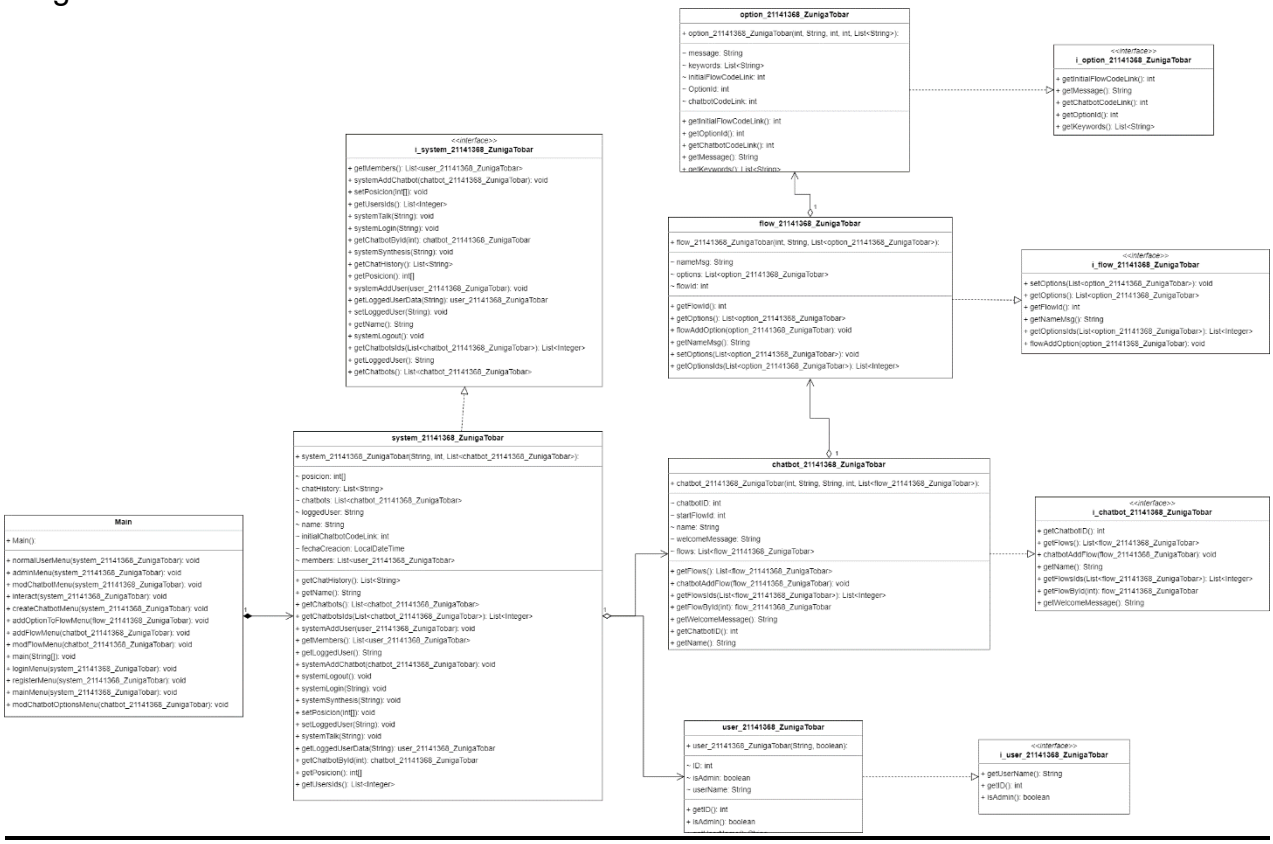
1. Tabla de funciones obligatorias

Tipo de Método	Nombre de Método	Funcionamiento
Constructor	option	Crea un objeto de clase Option, con los atributos
Constructor	flow	Crea un objeto de clase Flow,
Modificador	flowAddOption	Agrega una nueva opción al flujo, pero primero revisa si está repetido, si esto último se cumple, no hace nada.
Constructor	chatbot	Crea un objeto de clase chatbot, pero en el proceso de creación
Modificador	chatbotAddFlow	Agrega un nuevo flujo a un chatbot, pero primero revisa si está repetido, si esto último se cumple, no hace nada.
Constructor	system	Crea un objeto de clase
Modificador	systemAddChatbot	Agrega un nuevo chatbot al sistema, pero primero revisa si está repetido, si esto último se cumple, no hace nada.
Modificador	systemAddUser	Primero verifica comparando la ID del usuario de los demás miembros registrados, luego actualiza la lista de usuarios con el nuevo usuario agregado.
Modificador	systemLogin	Verifica si el usuario de entrada está registrado para después actualizar el atributo de “loggedUser” y al iniciar sesión se verifica si el usuario logeado es uno normal o un administrador, lo cual lo envía a sus correspondientes menús.
Modificador	systemLogout	Borra el historial del usuario logeado en ese momento, es decir, dejar el atributo chatHistory como una lista vacía y después deja vacío el atributo “loggedUser”.
Ninguno	systemTalk	Verifica si el mensaje de entrada pertenece a alguna opción o una palabra clave, si esto se cumple, llama a actualizar la posición e imprime el nuevo flujo dirigido y sus opciones.
Ninguno	systemSynthesis	Se supone que retorna una síntesis para un usuario según el historial
Las funciones obligatorias que no aparezcan en esta tabla, es porque no fueron definidas hasta ahora.		

2. Diagrama de análisis:



3. Diagrama de diseño:



4. TDAs Implementados:

4.1. TDA-system:

Tipo	Método	Descripción
Selector	getName	Obtiene el nombre del sistema
Selector	getLoggedUser	Obtiene el nombre del usuario que ha iniciado sesión
Modificador	setLoggedUser	Asigna el nombre de usuario que ha iniciado sesión
Selector	getPosicion	Obtiene una lista de 2 elementos para guiar al programa sobre a dónde va la interacción entre el usuario y un chatbot.
Modificador	setPosicion	Asigna una nueva posición después de una interacción entre usuario y un chatbot.
Selector	getChatHistory	Obtiene el historial de interacción entre un usuario y un chatbot
Selector	getLoggedUserData	Obtiene el objeto usuario deseado mediate el nombre de usuario
Selector	getMembers	Obtiene los usuarios registrados en el sistema
Selector	getChatbotById	Obtiene un chatbot mediante la id de este
Selector	getUsersIds	Obtiene una lista de las IDs de los usuarios registrados.
Selector	getChatbots	Obtiene los chatbots del sistema
Selector	getChatbotsIds	Obtiene las Ids de los chatbots del sistema.

4.2. TDA-option:

Tipo	Método	Descripción
Selector	getOptionId	Obtiene la ID de una opción
Selector	getMessage	Obtiene el mensaje de una opción
Selector	getChatbotCodeLink	Obtiene la id del chatbot ligado a la opción
Selector	getInitialFlowCodeLink	Obtiene la id del flujo ligado a la opción
Selector	getKeywords	Obtiene las palabras clave de una opción

4.3. TDA-flow:

Tipo	Método	Descripción
Selector	getNameMsg	Obtiene el mensaje de un flujo
Selector	getFlowId	Obtiene la id de un flujo
Selector	getOptions	Obtiene las opciones de un flujo
Modificador	setOptions	Asigna opciones a un flujo
Selector	getOptionsIds	Obtiene las IDs de las opciones de un flujo

4.4. TDA-user:

Tipo	Predicado	Descripción
Constructor	user_21141368_ZunigaTobar	Se crea un objeto de clase User donde la id se obtiene sacando el número del nombre de usuario
Selector	getUserId	Obtiene el ID de un usuario
Selector	getUserName	Obtiene el nombre del usuario.
Pertenencia	isAdmin	Retorna True o False si el usuario es administrador o no.

4.5. TDA-chatbot:

Tipo	Método	Descripción
Selector	getName	Obtiene el nombre de un chatbot.
Selector	getChatbotID	Obtiene la ID de un chatbot.
Selector	getWelcomeMessage	Obtiene el mensaje de bienvenida de un chatbot
Selector	getFlows	Obtiene los flujos de un chatbot
Selector	getFlowIds	Obtiene las ids de los flujos de un chatbot
Selector	getFlowById	Obtiene un flujo mediante su id

5. Autoevaluación requerimientos no funcionales:

Requerimientos No Funcionales	Puntaje
Autoevaluación	1
Lenguaje y herramientas de trabajo	1
Versión 11 de OpenJDK	1
Standard	1
Documentación JavaDoc	1
Instrucciones con el programa	1
Uso del paradigma	1
Organización	1
Historial	1
Diagrama de análisis	1
Diagrama de diseño	1
Prerrequisitos	0.75 No se definió la función system-simulate

6. Autoevaluación requerimientos funcionales:

Requerimientos Funcionales	Puntaje
TDA's	1
option	1
flow	1
flow-add-option	1
chatbot	1
chatbot-add-flow	1
system	1
system-add-chatbot	1
system-add-user	1
system-login	1
system-logout	1
system-talk	0.75 No entrega un mensaje final al terminar de interactuar con los chatbots
system-synthesis	0.25 solo entrega el historial del usuario
system-simulate	0 no se definió la función