DEPARTAMENTO DE
INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

Paradigmas de Programación

Proyecto Semestral de Laboratorio:

Paradigma Orientado a Objetos

Autor: Sebastián Paillao

Profesor: Edmundo Leiva

Fecha de entrega: 11 de diciembre, 2023

Índice

1-. Introducción

El presente informe se enmarca dentro del curso "Paradigmas de Programación", donde se propone un proyecto semestral de laboratorio que requiere ser desarrollado en diversos lenguajes de programación y siguiendo diferentes paradigmas. En esta entrega el foco será el paradigma orientado a objetos, utilizando el lenguaje de programación Java. Este paradigma se distingue por su enfoque en la creación y manipulación de objetos, lo que lo hace particularmente adecuado para proyectos complejos y estructurados, como el que se abordará a continuación.

Sobre la estructura del informe, éste posee la presente introducción con sus subsecciones para describir el problema, una descripción del paradigma con sus conceptos relacionados y los objetivos a plantear para el desarrollo de la entrega. Posteriormente se tiene el desarrollo junto a sus subsecciones análisis del problema, diseño de la solución, aspectos de implementación, instrucciones y resultados y autoevaluación final sobre el desarrollo de un sistema de chatbots.

Un chatbot es un software programado para simular interacciones conversacionales con usuarios humanos, principalmente a través de plataformas digitales como sitios web, aplicaciones de mensajería y redes sociales. Puede ser avanzado, utilizando inteligencia artificial para respuestas más complejas y adaptativas a la necesidad de cada usuario, o simple, basado en reglas predefinidas para respuestas específicas, limitaciones que tendrá el sistema a desarrollar.

Además, los chatbots se han convertido en herramientas esenciales en múltiples áreas, especialmente en el servicio al cliente y asistencia en línea. En estos contextos, ofrecen una disponibilidad constante, reduciendo los tiempos de espera para los usuarios y mejorando la eficiencia del servicio. También se utilizan en áreas como la educación, donde pueden facilitar el aprendizaje interactivo, y en el comercio electrónico, asistiendo a los clientes en la selección y compra de productos.

En el marco del proyecto de programación, el desarrollo de un chatbot en el paradigma orientado a objetos implica diseñar una arquitectura donde cada componente del chatbot, como el análisis de la entrada del usuario, la estructura de un chatbot y el manejo del estado de la conversación, se representa como un objeto. Esto proporciona una estructura clara y modular, permitiendo un desarrollo y mantenimiento más eficientes del sistema[1].

1.1-. Descripción del problema

El proyecto aborda el desarrollo de un sistema destinado a la creación, implementación y gestión de chatbots de tipo ITR (Respuesta de Interacción a Texto). Este tipo de chatbots se caracteriza por su estructura definida, respondiendo a consultas de los usuarios mediante opciones preestablecidas, sin procesar lenguaje natural de manera compleja ni emplear inteligencia artificial.

El sistema permitirá a los usuarios registrarse y participar activamente, interactuando con diversos chatbots que forman parte del entorno. Estas interacciones se basarán en respuestas textuales claras y concisas por parte del chatbot. Un aspecto clave del sistema es la capacidad de interconexión entre diferentes chatbots, lo que posibilita un flujo de diálogo más amplio y adaptado a las necesidades de los usuarios. Además, el sistema dará la opción

a los usuarios de obtener un historial de sus interacciones, mejorando la experiencia y permitiendo un seguimiento de sus consultas y respuestas recibidas.

Estos chatbot deben tener sus opciones y flujos definidos, dándole al usuario la posibilidad de interactuar con ellos y recorrer los flujos disponibles. Adicionalmente se podrá agregar flujos, opciones o chatbots al sistema, que aporten a la diversidad de las interacciones y aumenten las posibilidades y variedad de respuestas que el usuario podría obtener.

1.2-. Descripción del Paradigma:

El paradigma orientado a objetos (POO) se centra en el diseño de software alrededor de 'objetos', que son instancias de 'clases'. Las clases definen las propiedades y comportamientos (métodos) de los objetos. En POO, una 'interfaz' es un contrato que define un conjunto de métodos que una clase debe implementar. La 'herencia' permite a una clase derivar propiedades y comportamientos de otra, promoviendo la reutilización de código. El 'polimorfismo' permite a objetos de diferentes clases ser tratados como instancias de una clase común, facilitando la flexibilidad y extensibilidad del código. Estos conceptos son fundamentales para poder comprender el funcionamiento de un programa en POO:

Clases: En POO, una clase es una definición abstracta que representa un conjunto de propiedades y métodos comunes a un tipo de objeto. La clase actúa como una plantilla para crear instancias de objetos, especificando qué atributos (datos) y comportamientos (métodos) tendrán esos objetos.

Objetos: Son instancias concretas creadas a partir de una clase. Cada objeto tiene un estado único definido por los valores de sus atributos y puede realizar acciones a través de sus métodos. Los objetos son la base de la interacción en POO, que representan entidades del mundo real o conceptuales.

Métodos: Son funciones o procedimientos definidos dentro de una clase. Éstos describen los comportamientos o acciones que pueden realizar los objetos de esa clase. Los métodos pueden manipular los datos internos de un objeto (sus atributos) y proporcionar mecanismos para interactuar con otros objetos.

Herencia: Permite que una clase (conocida como subclase o clase hija) adquiera las propiedades y métodos de otra clase (llamada superclase o clase padre). La herencia facilita la reutilización del código y la creación de jerarquías de clases, donde las subclases pueden personalizar o ampliar las funcionalidades heredadas.

Polimorfismo: Es la capacidad de tratar objetos de diferentes subclases como si fueran de una clase común, lo que permite que un mismo método o interfaz funcione de manera diferente según el objeto que lo invoque. El polimorfismo aumenta la flexibilidad y la capacidad de generalización del código.

Interfaces: Son contratos que definen un conjunto de métodos (sin implementar) que las clases deben y se comprometen a implementar. Las interfaces permiten estandarizar y asegurar un conjunto de comportamientos en diferentes clases, ofreciendo una forma de garantizar que ciertas clases proporcionen funcionalidades específicas.[2]

1.3-. Objetivos

El objetivo del desarrollo de este proyecto es completar el aprendizaje del paradigma orientado a objetos, utilizando el lenguaje Java. Para programar en este lenguaje se utilizará el IDE IntelliJ, que facilita el trabajo al momento de estar creando código y también con en manejo de las diversas clases que se utilizarán. Este aprendizaje se llevará a cabo desarrollando las soluciones a las diversas problemáticas planteadas en el enunciado del proyecto sobre la creación de un sistema de chatbots, que se espera concluir en su totalidad logrando un funcionamiento perfecto en función de los requerimientos establecidos.

2.- Desarrollo

En esta sección se plasmará el proceso del desarrollo de la solución para el problema, desde el análisis de éste mismo hasta los resultados finales.

2.1.- Análisis del problema

Como requerimientos funcionales, se deben crear las **clases** necesarias para poder implementar los métodos requeridos, cada clase debe estar estructurada en archivos separados de las otras existentes, para luego ser llamada cada una según sus métodos dentro del archivo 'main' al momento de compilar el programa. Este programa debe permitir al usuario interactuar con el sistema o modificarlo a través de un menú. Las clases mínimas requeridas, además de sus métodos solicitados son las siguientes:

Clase: Option

Constructor objeto 'option', debe tener como argumentos las siguientes variables:

code (int) x *message* (String) x *chatbotCodeLink* (int) x *initialFlowCodeLink* (int) x *keywords* (List).

Clase: Flow

Constructor objeto 'flow', debe tener como entrada las siguientes variables:

ID (int) x *message* (String) x *options* (List).

Modificador 'flowAddOption': Agrega una opción a un flujo, verificando que no pertenezca ya al flujo y retornando false si ya está incluida. Debe tener como entrada la siguiente variable:

option (Option).

Clase: Chatbot

Constructor objeto 'chatbot', si dentro de los flujos hay duplicidad, agrega sólo uno de los duplicados. Debe tener como entrada las siguientes variables:

chatbotID (int) x *name* (String) *x welcomeMessage* (String), *startFlowId* (int), *flows* (List).

Modificador 'chatbotAddFlow': Agrega un flujo a un chatbot, verificando que no pertenezca ya al chatbot y retornando false si ya está incluido. Debe tener como entrada la siguiente variable:

flow (Flow).

Clase: User

Constructor objeto ‘user, si dentro del sistema ya existe un usuario con este nombre, no se agregará. Debe tener como entrada la siguiente variable:

username (String).

Clase: System

Constructor objeto ‘system’ debe tener como entrada las siguientes variables:

name (String) x *initialChatbotCodeLink* (int) x *chatbots* (List).

Modificador ‘systemAddChatbot’ Agrega un chatbot al sistema, retornando false si ya está incluido. Debe tener como entrada la siguiente variable:

chatbot (Chatbot).

Modificador ‘systemAddUser’ Registra un nuevo usuario en el sistema, retornando false si ya hay un usuario con el mismo nombre. Debe tener como entrada la siguiente variable:

user (User).

Método ‘systemLogin’ Permite al usuario iniciar sesión, retornando false si ya hay una sesión iniciada por otro usuario o si el usuario no existe. Debe tener como entrada la siguiente variable:

username (String).

Método ‘systemLogout’ Cierra la sesión del usuario actual después de verificar que haya un usuario con su sesión iniciada. No tiene variables de entrada.

Método ‘systemTalk’, permite al usuario interactuar con el sistema, ingresando una opción numérica o una palabra clave para encontrar la opción que la contiene, en caso de no coincidir con ninguna opción retornar false. Debe tener como entrada las siguientes variables:

message (String)

Método ‘systemSynthesis’, imprime en pantalla el historial de interacciones entre cierto usuario y el sistema de chatbots, contenido en chatHistory y formateado para mejor experiencia de usuario. Debe tener como entrada la siguiente variable:

username (String).

Método ‘systemSimulate’, debe permitir simular interacciones entre dos chatbots del sistema definiendo un máximo de interacciones, debe utilizarse el método ‘myRandom’ para generar números aleatorios que definirán los caminos que tomará esta conversación. Debe tener como entrada las siguientes variables:

maxInteractions (int) x seed (int) x *system* (System).

Además debe plantearse la clase chatHistory junto a sus constructores y predicados necesarios. Finalmente, debe construirse un menú que permita las interacciones y le de opciones de acciones al usuario para manejar el programa y su correcto funcionamiento, menú que será ejecutado desde el archivo *main* y compilado mediante Gradle.

2.2.- Diseño de la solución

Antes de comenzar a escribir el código del programa requerido, se debe planificar la estructura y el funcionamiento de la solución que se llevará a cabo. Para esto se plantea el diagrama UML de análisis (ver Anexo 1), en el cual se representa cada uno de las clases y sus relacione. Cada clase tiene su estructura y algunos son agregadores de otros, en el diagrama, por ejemplo, puede observarse que chatbot y user son agregados a la clase system, ya que los sistemas a crear contendrán a los chatbots y usuarios, además de los otros parámetros y símbolos necesarios para establecer las relaciones entre el resto de las clases. Puede observarse también en el diagrama que los chatbots son agregadores de flujos, y los flujos a su vez agregan opciones. Además, el user, que es contenido por el sistema, también contendrá dentro de su estructura a su historial de interacciones con el sistema, necesario para completar la funcionalidad de los predicados requeridos.

Se plantea además la solución particular de cada uno de los requerimientos funcionales:

Clase: Option

> **Representación:** Option es un objeto que contendrá en su estructura el código de la opción, el mensaje, el ChatbotCodeLink, el InitialFlowCodeLink y una lista de keywords o palabras clave relacionadas a esta opción.
>
> **Constructor**: Se ingresan los datos requeridos, keywords también puede ser una lista vacía, retorna una lista con los elementos ingresados y la estructura propuesta.

El código de las opciones es utilizado para comprobar duplicidad dentro de un flujo al hacer uso del modificador *flowAddOption,* utilizado cuando el usuario elige la opcion en el menú de crear y agregar una opción a un flujo.

Clase: Flow

> **Representación:** Flow es un objeto que contendrá en su estructura el ID del flujo, su mensaje y una lista de opciones.
>
> **Constructor**: Se ingresan los datos requeridos, la lista de opciones también puede estar vacía, retorna una lista con los elementos ingresados y la estructura propuesta.

El modificador ***flowAddOption*** utilizará esta estructura de *Flow* para obtener las opciones del flujo y a su vez obtener los códigos de éstas, con el fin de no agregar ninguna opción que comparta código con alguna de las pertenecientes al flujo y retornar *False* en ese caso. En caso contrario, agregará a la opción al flujo. Este método sólo será llamado inmediatamente después de crear la opción, mediante el menú de administrador.

Clase: Chatbot

> **Representación:** Chatbot es una lista que contendrá en su estructura el ID del chatbot, su nombre, un mensaje de bienvenida, su flujo inicial y una lista de flujos.

Constructor Se ingresan los datos requeridos, la lista de flujos también puede estar vacía, retornará una lista con los elementos ingresados y la estructura propuesta.

El modificador ***chatbotAddFlow*** utilizará esta estructura de *Chatbot* para obtener los flujos del chatbot y a su vez obtener los ID de éstos, con el fin de no agregar ningún flujo que comparta ID con alguno de los pertenecientes al chatbot y retornar *False* en ese caso. En caso contrario, agregará el flujo al chatbot. Este método sólo será llamado inmediatamente después de crear el flujo, mediante el menú de administrador.

Clase: User

Representación: User es un objeto que contendrá un nombre de usuario, un historial de chat y un estado de sesión.

Constructor: Se ingresa dentro de *systemAddUser* el nombre del usuario a agregar y se llamará a *user* para crear una lista *User* que contiene el *username*, un historial de chat vacío, un estado de sesión 'notLogged' y unos permisos de usuario 'default', listos para modificarse según lo que el usuario elija al momento de llamar a *systemAddUser* dentro del menú.

Clase: ChatHistory

Representación: ChatHistory es un objeto que contendrá la marca de tiempo, el usuario al cual pertenece el historial, el mensaje enviado por el usuario, el mensaje de la opción elegida según la entrada, el nombre del chatbot actual y la respuesta del sistema como un objeto flujo, que mostrará las nuevas opciones a elegir.

Constructor: Se ingresarán los datos de cada argumento por medio de systemTalk.

Clase: System

Representación: System es un objeto que contendrá el nombre del sistema, el InitialChatbotCodeLink, una lista de chatbots, una lista de usuarios, un flujo actual y un chatbot actual, que trabajarán como el estado de las interacciones.

Constructor: Se ingresan todas las variables requeridas exceptuando la lista de usuarios, y lo que se creará dentro del mismo constructor y se inicializará como lista vacía para contener a futuro a los distintos usuarios a registrar, retornará entonces una lista con toda la estructura mencionada en su representación.

El modificador ***systemAddChatbot*** utilizará esta estructura de *system* para obtener los chatbots del sistema y a su vez obtener los ID de éstos con el fin de no agregar ningún chatbot que comparta ID con alguno de los pertenecientes al sistema y retornar *False* en ese caso. En caso contrario, agregará al o a los chatbots al sistema. Este método sólo será llamado inmediatamente después de crear el chatbot, mediante el menú de administrador.

El modificador ***systemAddUser*** utilizará la lista de *Users* del sistema para obtener el nombre de usuario de cada uno de los usuarios, con el fin de evitar crear un nuevo usuario con el mismo username de otro, retornando *False* en ese caso. Si el nombre de usuario ingresado no está en la lista de usuarios, se agregará a ésta, generando también dentro de este user un ChatHistory, un estado de 'notLogged', manifestando que su sesión aún no ha sido

iniciada, y unos permisos de usuario 'default', el menú. Este método sólo será llamado inmediatamente después de crear al usuario, mediante el menú de administrador.

El método ***systemLogin*** utilizará la lista de *users* del sistema para consultar si hay algún usuario con su sesión iniciada, si el estado de sesión de uno de los usuarios está establecido como 'logged', el programa retornará *false*. En caso contrario, se cambiará el estado del usuario al cual corresponde el username ingresado de 'notLogged' a 'logged' dejando al actual usuario como currentUser en el sistema. Este método sólo será llamado mediante el menú de usuario.

El método ***systemLogOut*** utilizará la lista de *users* del sistema para consultar si hay algún usuario con su sesión iniciada, si el estado de alguno de los usuarios está establecido como 'logged', se cambiará a 'notLogged' y el currentUser del sistema será 'null, en caso de que ningún usuario esté en estado de sesión iniciada, el programa retornará *False*. Este método sólo será llamado mediante el menú de usuario.

El método ***systemTalk*** utilizará al *system* para obtener los chatbots de los chatbots sus flujos, de los flujos sus opciones y también de la lista de usuarios usará el historial y el estado de sesión. Primero verificará que haya un usuario con su estado de sesión en 'logged', de lo contrario, retornará *False,* después comparará la entrada del usuario con el código de las opciones del flujo del chatbot actual, al encontrar coincidencia entre el código de las opciones o las keywords y la entrada del usuario, dirigirá al usuario al próximo flujo a seguir actualizando el flujo y chatbot actuales y guardando en el ChatHistory la interacción con sus parámetros determinados. Este método sólo será llamado mediante el menú de usuario.

El método ***systemSynthesis***, utilizará el *chatHistory* del user que tiene su estado de sesión iniciado para imprimir en pantalla las interacciones del usuario en un formato establecido para ser más entendible y agradable a la vista por el usuario. Será llamado a través del menú de usuario.

Clase: Menu

Representación: El menú contendrá el único sistema creado, manejará al usuario que tenga su sesión iniciada y procesará las entradas. Se usarán switch case para manejar los diversos casos y opciones elegidas por el usuario.

Constructor: Se inicializará dentro del *main*, después de haber creado el system base, teniéndolo como argumento.

Clase: Main

Representación: Será donde se instancien las opciones, flujos, chatbots y el sistema, además de llamar al menú con el cual el usuario interactuaría.

2.3.- Aspectos de implementación

Para el desarrollo de esta problemática se hizo uso del editor de texto y compilador IntelliJ, utilizado por recomendación del profesor. Se utilizó el lenguaje de programación Java, usando OpenJDK versión 11 con Gradle Wrapper y se hizo uso sólo de funciones nativas del lenguaje mencionado y sin incluir bibliotecas externas.

Como pudo plantearse en el diagrama (ver Anexo 1) y en el diseño de la solución, el programa utiliza clases relacionadas contenidas unas dentro de otras para mantener una estructura sólida y accesible utilizando los métodos y selectores establecidos.

Dentro de la carpeta a la cual pertenece el informe actual se podrá encontrar la carpeta de código fuente, dentro de la cual se plantea cada clase en un archivo distinto, todas siendo utilizadas por sus métodos junto a la clase menú en el archivo *main* para funcionar en conjunto con las instancias creadas en el archivo mencionado.

2.4.- Instrucciones de uso

Manual de usuario:

Para comenzar debe ejecutarse el proyecto por medo de consola utilizando .\gradlew run o por medio del IDE, que permite la ejecución rápida. Al ejecutar el programa se mostrarán las siguientes opciones:

Bienvenido al Sistema de Chatbot

1. Iniciar Sesión
2. Registrar Usuario
0. Salir

Para iniciar sesión en una cuenta ya creada sólo debe ingresar en consola la opción “1” e ingresar el nombre de usuario. Si desea testear los permisos y facultades de un administrador, como usuario debe ingresar “admin”. Lo que le permitirá iniciar sesión como un administrador instanciado en el main.

Para registrar un usuario nuevo debe ingresar en consola el número “2”, ingresar el nombre de usuario que quiera usar y seleccionar los permisos que desee que tenga esa cuenta.

Ingrese un nombre de usuario para el registro: sebastian

Seleccione el tipo de usuario:
1. Usuario Normal
2. Usuario Administrador
Seleccione una opción: 1

Usuario creado con éxito:
Nombre de usuario: sebastian
Permisos: default

Permisos de usuario normal y administrador:
Los usuarios que tengan permisos de **usuario normal** sólo tendrán disponibles las siguientes opciones:

Opciones de Usuario:
1. Interactuar con el chatbot
2. Consultar mi historial

3. Simular diálogo entre chatbots
0. Cerrar sesion (ingresar 0 dos veces)

La primera le llevará a interactuar con el sistema mediante texto, el usuario puede ingresar tanto el número relacionado a la opción como una palabra clave relacionada a alguna de ellas.

La segunda opción le imprimirá en pantalla su historial de interacciones en caso de que ya haya interactuado con el sistema.

La tercera opcion de simular diálogo no fue implementada, así que sólo imprimirá un gatito con un mensaje al ser seleccionada.

Los usuarios con **permisos de administrador** tendrán disponibles todas las opciones del usuario normal, además de las siguientes opciones extra:

4. Crear y Agregar Chatbot al sistema
5. Crear y Agregar flujos a un Chatbot
6. Crear y Agregar opciones a un flujo

Cada una de estas opciones pedirá los argumentos constructores de cada tipo de clase según su contexto uno por uno y además se agregarán al chatbot o flujo del cual se ingrese la ID. En el caso de la opción 4 no se pedirá sistema al cual agregar el chatbot ya que el sistema es único. No se pueden ingresar opciones, flujos ni chatbots que posean una ID que pertenezca a un objeto de su tipo ya instanciado. Se pueden revisar los objetos instanciados en el archivo *main* para evitar inconvenientes con estas opciones.

Para cerrar sesión sólo se debe estar en el menú principal del usuario e ingresar el número 0 dos veces seguidas para luego elegir la opción de cerrar sesión.

2.5.- Resultados y Autoevaluación

Los resultados obtenidos son en su mayoría positivos, exceptuando la implementación del último método systemSimulate, que no fue implementado. El resto de los requerimientos funcionales y constructores de clases fueron implementados, como puede verse en el diagrama final (ver Anexo 2) y funcionan de manera perfecta según lo estipulado en el 100% de los casos, se hicieron las pruebas en el *main* y también a través de interacciones en el menú para evitar duplicidad en objetos, con resultados positivos. De las 14 funciones planteadas se logró desarrollar 13 a la perfección, la autoevaluación resulta con un puntaje de 13 de los 14 puntos planteados.

3.- Conclusiones

Se puede concluir que se cumplieron en su mayoría los objetivos establecidos, se logró desarrollar el aprendizaje de uso y manejo del paradigma orientado a objetos mediante el lenguaje Java. Se aprendió sobre el funcionamiento y estructuras de este paradigma llevando a cabo el desarrollo del código y finalizándolo, cumpliendo con la gran mayoría de los objetivos planteados. Se tuvo problemas con la implementación de la últimas funcion debido al tiempo disponible, pero se espera implementar para que complete sus funciones en un

futuro. Con el paradigma orientado a objetos se pueden instanciar objetos y manipularlos a gusto además de relacionarlos entre ellos, una libertad que no era existente en los otros paradigmas estudiados en el ramo, tema que facilitó mucho el trabajo y la aplicación de lo aprendido en clases.

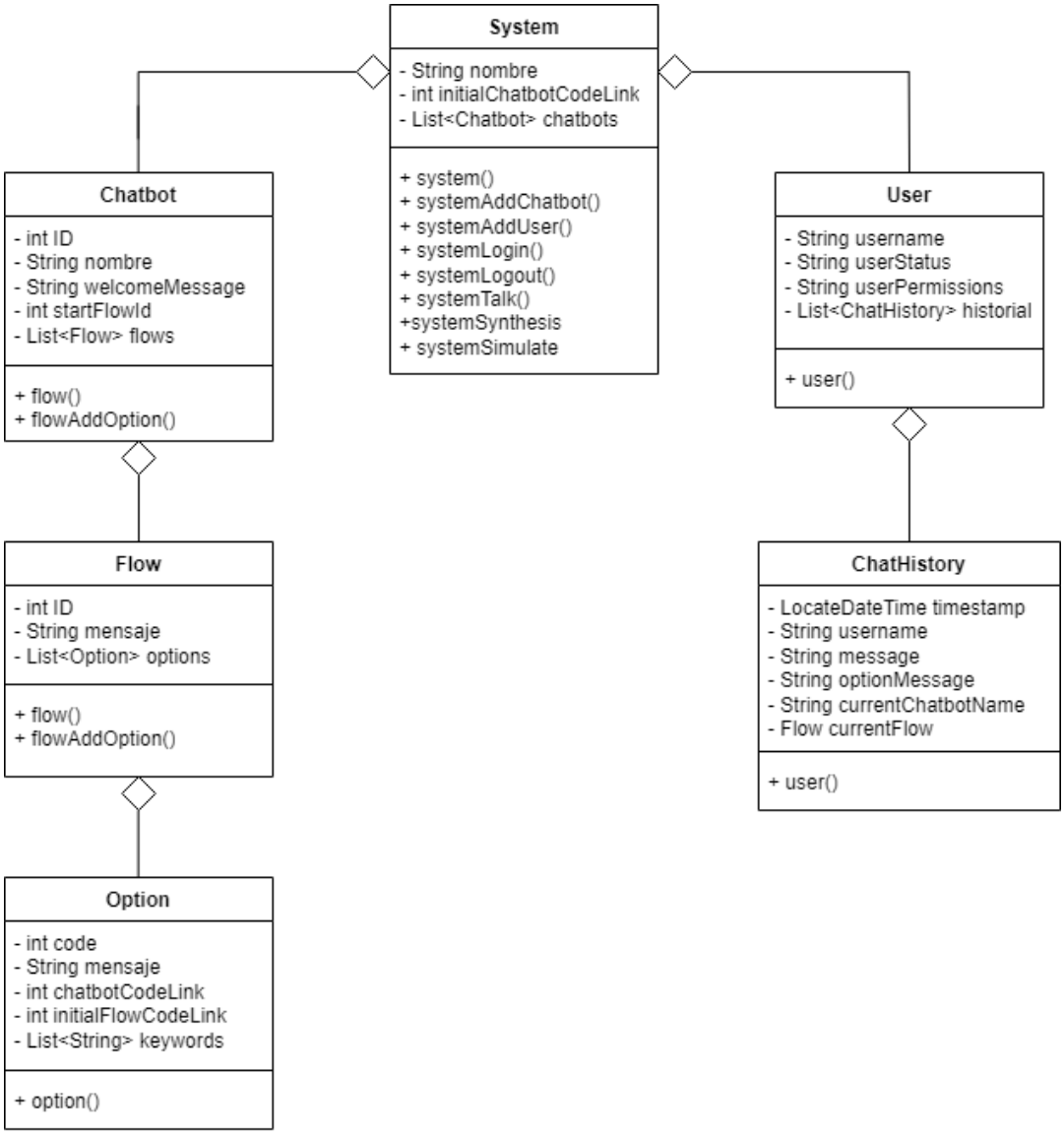
4.- Referencias en APA

1.- Birkenstock, R. (2023) Tecnología de Chatbots: Pasado, Presente y Futuro. Recuperado de https://www.toptal.com/insights/innovation/chatbot-technology-past-present-future.

2- Oracle. (s. f.). Documentación de Java. Recuperado de https://docs.oracle.com/en/java/.

5.- Anexos

1.- Diagrama de análisis.



2- Diagrama Final