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

Laboratorio 3 - Paradigma Orientado a Objetos
Sistema de chatbots simplificado
Java

Nombre: Vicente Gerardo Arce Palacios
Curso: Paradigmas de Programación
Sección A-1
Profesor: Roberto González Ibañez

11 de Diciembre de 2023

Tabla de contenidos

1. Introducción

En el presente informe se abordará el problema propuesto en el enunciado del laboratorio tres del ramo "Paradigmas de Programación". En este último laboratorio se tiene como requisito el uso del Paradigma Orientado a Objetos (POO), en donde se utilizó Java como lenguaje de programación. El informe se separara en una breve descripción del problema y del paradigma a utilizar, más adelante en el desarrollo se detallara el análisis del problema, el diseño de la solución, algunos detalles en la implementación junto con las instrucciones de uso, se darán los resultados y una autoevaluación para finalmente dar una conclusión al proyecto.

1.1. Descripción del problema

Se presenta como desafió el desarrollo de un programa usando el Paradigma Orientado a Objetos en el lenguaje Java, dicho programa se trata de una simplificación de un sistema de chatbots, con el cual se podrá interactuar de distintas maneras. Uno de los primeros problemas que se presentan en este laboratorio es el tener que manejar dos tipos de usuarios (administrador y común), los cuales tienen la capacidad de interactuar de distintas maneras con el sistema de chatbots. El usuario común únicamente puede interactuar con el sistema, es decir, puede loguearse, desloguearse, hablar con el sistema, ver su historial y finalmente realizar una simulación con el sistema. El usuario administrador puede realizar lo mismo que el usuario común y además tiene la capacidad de crear opciones, flujos, chatbots y sistemas, también puede modificar los anteriores objetos. El problema de todo lo nombrado con anterioridad radica en como se puede bifurcar las operaciones que cada tipo usuario puede realizar. Por otro lado, otro problema que se presenta es como representar cada clase, que atributos deben considerar para poder suplir todos los requisitos funcionales y entre otros. El como se solucionaron estos problemas se detallan en el análisis y diseño de la solución.

1.2. Descripción del paradigma

El Paradigma Orientado a Objetos, como su nombre lo indica, se basa en la definición de objetos, estos son abstracciones de entidades los cuales pueden tener atributos y métodos, esto con la finalidad de que los objetos interactúen entre si. Cabe destacar que el POO es un paradigma imperativo, es decir, se basan en el "COMO" llegamos al "QUE".
Algunos de los elementos más importantes del POO se nombran a continuación. **Clases**: definición de las características de objetos las cuales incluyen atributos y métodos. Una clase es una implementación de un TDA. **Objetos**: instancias de una clase. **Atributos**: corresponde a lo que compone una clase. Son las características que diferencian a un objeto de otro. **Métodos**: expresan los comportamientos que pueden realizar un objeto sobre si mismo o sobre otros objetos. **Polimorfismo**: capacidad de un objeto de tomar diferentes formas. Existen dos tipos principales de polimorfismo, esta la sobrecarga y la sobrescritura. **Herencia**: permite que una clase (hija) herede los atributos y métodos de otra clase (padre).

2. Desarrollo

En este apartado se abordara el análisis del problema, el diseño de la solución, aspectos en la implementación, las instrucciones de uso junto con los resultados y una autoevaluación.

2.1. Análisis del problema

Primeramente se considerara un **menú**, este es la vía por donde el usuario tiene la posibilidad de interactuar con el programa. El menú consiste en un conjunto de funcionalidades, con este menú podrá interactuar un tipo de usuario común y/o uno administrador.
Existirá una clase abstracta llamada **AbstractUser**, la cual como atributos tiene username, historial e isAdmin. Esta clase tendrá como hijas las clases **ComunUser** y **AdminUser**. La clase ComunUser siempre construirá el usuario con su atributo isAdmin en false, mientras que AdminUser lo construirá con un true. Lo anterior se analizó de tal manera que en el menú se controle con una sentencia if la lógica de si el usuario es común o administrador.
Como se indico en la descripción del problema, se deben abordar las distintas funcionalidades con las que los usuarios pueden interactuar, es por ello que se consideran las clases fundamentales: **Option**, **Flow**, **Chatbot**, **System**, **Menu**, **Main** y las ya mencionadas **AbstractUser** y sus clases hijas.
Abordando la clase System, esta contiene chatbots, chatbots que contienen flujos y flujos que contienen opciones. La clase System se compone por su fecha de creación, el nombre, el código del chatbot inicial del sistema, la lista de chatbots, la lista de usuarios registrados, el username del usuario logueado y los códigos actualChatbotCodeLink, actualFlowCodeLink y placeHolderSimulate. Habiéndose analizado el problema, se considera que con los atributos ya nombrados es suficiente para resolver el problema, ya que esta la lista de usuarios, la cual permitirá registrar usuarios en el sistema, el atributo que indica el username del usuario logueado, lo cual permite loguear usuarios y desloguearlos. Los códigos actualChatbotCodeLink, actualFlowCodeLink permitirán la realización de el requisito funcional systemTalk lo cual implica que se podrá generar un historial y con ello realizar el requisito funcional systemSynthesis, finalmente el atributo placeHolderSimulate sera de gran utilidad para la realización del requisito systemSimulate.
El Menu es otra de las clases importantes, ya que esta tiene todos los métodos que manejan la lógica del menú con el que el usuario interactuara. Este tiene ciertos atributos tales como opcionesMain, flujosMain, chatbotsMain y sistemasMain. Los atributos mencionados se incluyeron en el menú con la finalidad de ir guardando las opciones, flujos, chatbots y sistemas que el usuario administrador vaya creando, esto con la finalidad de poder utilizarlos a futuro. Por ejemplo, si el usuario administrador quiere crear un flujo y previamente creo una opción, al momento de crear el flujo el menú pedirá al usuario las cantidad de opciones que quiere agregar al flujo, posteriormente se le mostrara al usuario las opciones que ha creado y este tendrá que elegirlas según su índice. La lógica anterior aplicaría al momento de querer modificar un flujo y modificar o crear un chatbot y un sistema.

2.2. Diseño de la solución

Para el diseño de la solución, se hicieron dos diagramas UML, uno de análisis y uno de diseño. El primer diagrama puede ser encontrado en el Anexo 1, este fue realizado antes de empezar el desarrollo del código. El segundo diagrama fue realizado tras haber implementado la solución, este se encuentra en el Anexo 2.

Las clases que se ven en el diagrama de clases son las que se consideraron fundamentales para la resolución del problema, a continuación se detallaran las clases:
Clase Option: Su representación esta por un código, un mensaje, un chatbotCodeLink, un initialFlowCodeLink y una lista de keywords.
Clase Flow: Su representación esta dada por un id, un nameMsg y una lista de options.
Clase Chatbot: Su representación esta dada por un chatbotid, un name, un welcomeMessage, un startFlowId y una lista de flows.
Clases de Usuarios: La clase padre "AbstractUser" esta representada por los atributos username, un historial y un isAdmin. Las clases "ComunUser" y "AdminUser" heredan los atributos y métodos de AbstractUser, esto con la diferencia de que el ComunUser se construirá con el atributo isAdmin en false y el AdminUser con dicho atributo en true, esto con la finalidad de diferenciarlos cuando se implemente el menú.
Clase System: Su representación esta dada por su fecha de creación, su nombre, un initialChatbotCodeLink, una lista de chatbots, una lista de usuarios abstractos (para que soporte usuarios comunes y administradores), un usuario logueado, un actualChatbotCodeLink (El cual manejara el código del chatbot en el que se encuentra el sistema), un actualFlowCodeLink (el cual maneja el código en el que que se encuentra el sistema, en un inicio tendrá como valor -1, esto para controlar casos bases) y finalmente un placeHolderSimulate (este controla si se ha iniciado una simulación o no).
Clase Menu: Su representación esta dado por una lista de opciones, una lista de flujos, una lista de chatbots y una lista de sistema, esta clase contiene los métodos necesarios para manejar la lógica de los menú y submenus del programa.
Clase Main: Esta clase es la que será ejecutada para dar inicio al sistema, esta simplemente crea un menú y se encarga de inicializarlo, a su vez, esta clase es la encargada de inicializar el sistema inicial.

Las clases mencionadas con anterioridad son las que se implementaron para poder solucionar el problema. El sistema fue diseñado con la finalidad de poder suplir los requisitos funcionales, es por esto mismo que se añadieron los atributos actualChatbotCodeLink, actualFlowCodeLink y placeHolderSimulate. Con esta forma de diseño del sistema se pudieron realizar todos los requisitos funcionales presentados en el enunciado. El menú se diseño de tal manera que los objetos que el usuario administrador vaya creando se guarden en listas para después presentárselas al usuario al momento de querer usarlos.

2.3. Aspectos de implementación

Para este proyecto se uso el IDE inteliJ IDEA en su versión 2023.2.5. En el caso de JDK, se utilizo la versión 11.0.21. Se usaron únicamente funcionalidades de la librería estándar de Java y el programa fue compilado y ejecutado en el sistema operativo Windows 11.
Este laboratorio, a diferencia de los anteriores, no cuenta con un script de pruebas, pero si cuenta con un sistema creado con anterioridad con el cual el usuario podrá interactuar si así lo desea.

2.4. Instrucciones de uso

Primeramente, asegúrese de contar con todos los archivos del proyecto en una misma carpeta, desde aquí usted tiene dos opciones. La primera opción es abrir el proyecto desde la terminal, luego ejecutar los siguientes comandos dependiendo de su sistema operativo, en Linux para compilar use el comando ./gradlew build y para ejecutar el comando ./gradlew run, en Windows para compilar use el comando gradlew.bat build y para ejecutar el comando gradlew.bat run. La segunda opción que se tiene es abrir el proyecto en un IDE como Intelij y ejecutar el archivo Main.
Como primera opción se le mostrara con que sistema quiere interactuar, para esto indique el índice de "Chatbots Paradigma", esto para que se empiece la interacción con el sistema precargado, luego ingrese la opción para registrarse, aquí se recomienda registrar un usuario común y uno administrador, luego selecciones volver y loguee uno de los usuarios que registro, se recomienda que primero se loguee con un usuario común para probar las funcionalidades como un usuario común y luego las del usuario administrador. Una vez se haya logueado, si se logueo como usuario común, tendrá la opción de deloguearse, hablar con el sistema, ver su historial, hacer una simulación con el sistema, realizar una simulación con el sistema, ver el sistema y salir (misma opción que desloguearse). Si se loguea como usuario administrador, tendrá las opciones que el usuario y común, pero además de ello podrá crear opciones, flujos, chatbots y sistemas, añadir una opción a un flujo, añadir un flujo a un chatbot y añadir un chatbot a un sistema, para estas ultimas tres funcionalidades se le pide haber creado con anterioridad un flujo, un chatbot y un sistema respectivamente. Si se crea un nuevo sistema y se quiere interactuar con este, debe volver hasta llegar al primer menú, en este menú además de presentarse el "Chatbots Paradigmas", se encontraran el o los nuevos sistemas que se hayan creado, solo basta que seleccione con cual desea interactuar.
Importante: Un fallo que se puede llegar a generar es si se hicieron modificaciones sobre el sistema precargado (como añadir un nuevo chatbot) o si se creo un nuevo sistema desde cero, la responsabilidad de si tienen sentido las modificaciones hechas recae en el usuario administrador, es decir, que hay una posibilidad de que algunas funcionalidades no funcionen a la perfección si se hace una modificación que carezca de sentido. Al hacer pruebas del programa no se encontraron más inconvenientes, funcionando correctamente todas las funcionalidades.

2.5. Resultados y autoevaluación

El grado de alcance de los requisitos funcionales fue logrado en su totalidad, contándose con todos los requerimientos funcionales y no funcionales. Se lograron los resultados previstos y algunos de los ejemplos de el funcionamiento del programa se pueden encontrar en el Anexo 3.
La autoevaluación puede ser encontrada en el Anexo 4, con un puntaje siguiendo el siguiente formato: 0: No realizado/ 0.25: Funciona el 25 % de las veces/ 0.5: Funciona el 50 % de las veces/ 0.75: Funciona el 75 % de las veces/ 1: Funciona el 100 % de las veces.

3. Conclusión

Luego de la realización del proyecto, se puede concluir que se cumplió el objetivo de implementar un sistema de chatbots ocupando el lenguaje Java, siguiendo únicamente el paradigma orientado a objetos, de este se adquirió el aprendizaje necesario sobre el paradigma, el cual en un inicio fue desafiante, sobretodo por la gran cantidad de conceptos que este aborda, pero a medida del avance del proyecto, fue bajando la dificultad debido a la comprensión que se tuvo del paradigma y por la costumbre de trabajar con el paradigma imperativo.
Haciendo la comparación con los anteriores laboratorios, este fue el más sencillo de llevar a código, pero una de las complicaciones que se tuvieron fue el tiempo, ya que este proyecto de laboratorio, a comparación de los anteriores, requirió bastante tiempo para la comprensión de este. Aún con las complicaciones, se llevaron a cabo todos los requisitos funcionales y no funcionales de gran manera, realizándose el proyecto por completo obteniéndose un programa completamente funcional.
Dejando lo anterior de lado, no hubo mayores complicaciones de compresión de conceptos o problemas con las herramientas como Git o Java.
Al ser este laboratorio el ultimo del ramo, se puede concluir que se logró en cada proyecto de laboratorio adentrarse en cada paradigma, ampliándose la forma de ver el como se puede llegar a resolver un mismo problema de diferentes maneras.

4. Bibliografía

Pérez, S. D. (s/f). Qué es la Programación Orientada a Objetos. Intelequia. https://intelequia.com/es/blog/post/qu

Mestras, J. P. (2007). Objetos y Clases. Ucm.es. https://www.fdi.ucm.es/profesor/jpavon/poo/1.1.objeto

Bernal , J. (2012). Programación orientada a Objetos con Java - UPM. Programación Orientada a Objetos con Java. https://www.etsisi.upm.es/sites/default/files/curso_2013_14/MASTER/MIW.JEE.POOJ.pdf

5. Anexos

5.1. Anexo 1: Diagrama de análisis

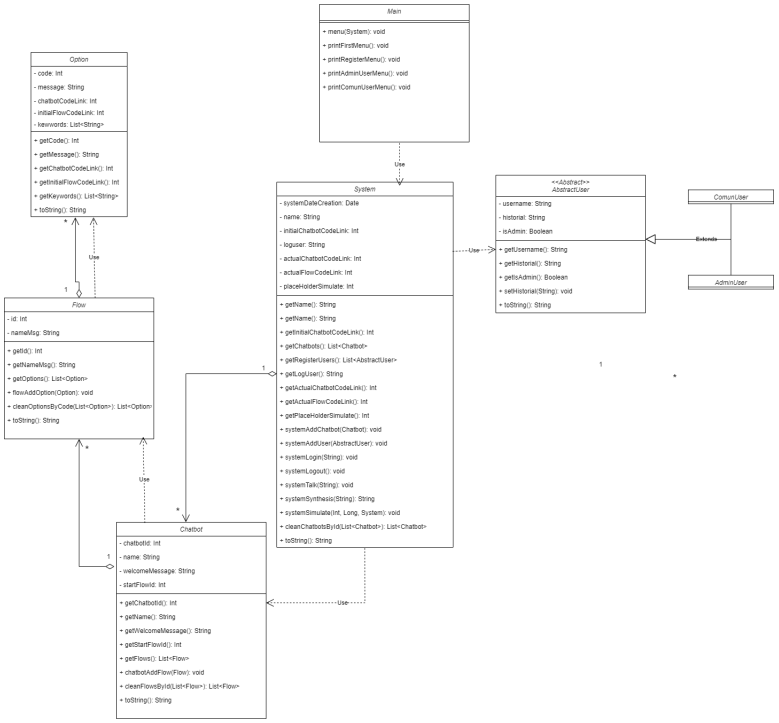


Figura 1: Diagrama de análisis UML.

5.2. Anexo 2: Diagrama de diseño

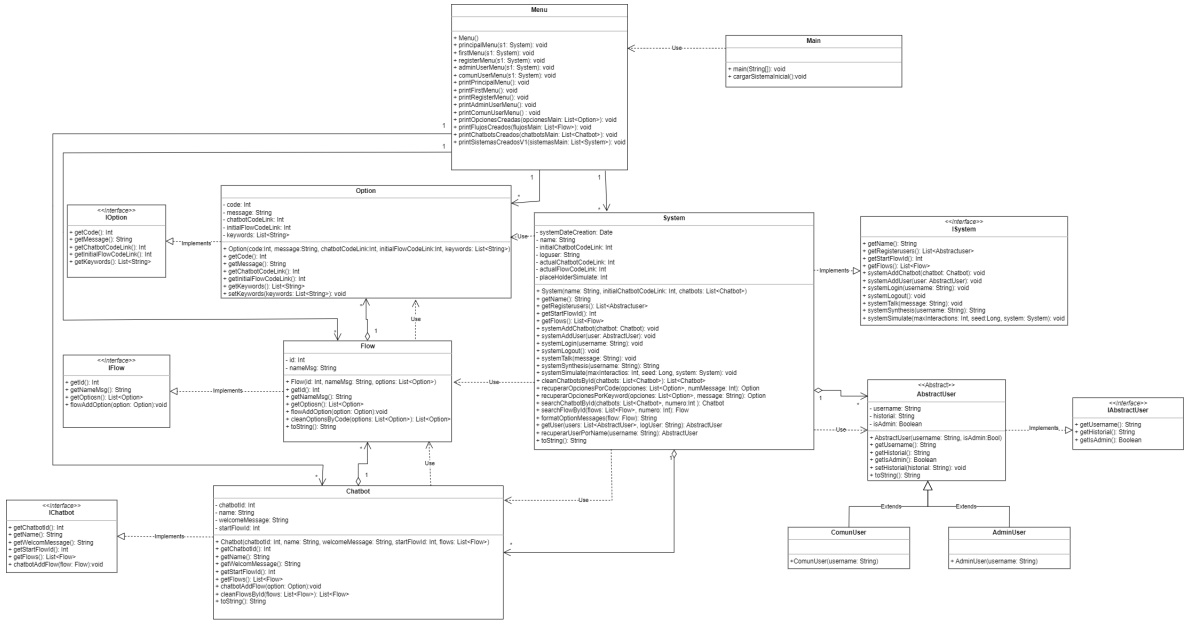


Figura 2: Diagrama de diseño.

5.3. Anexo 3: Ejemplos

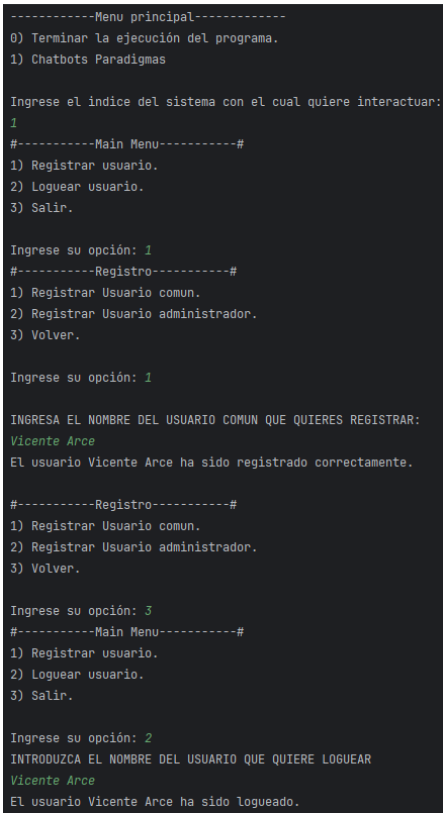


Figura 3: Seleccionando sistema y registrando usuario común.

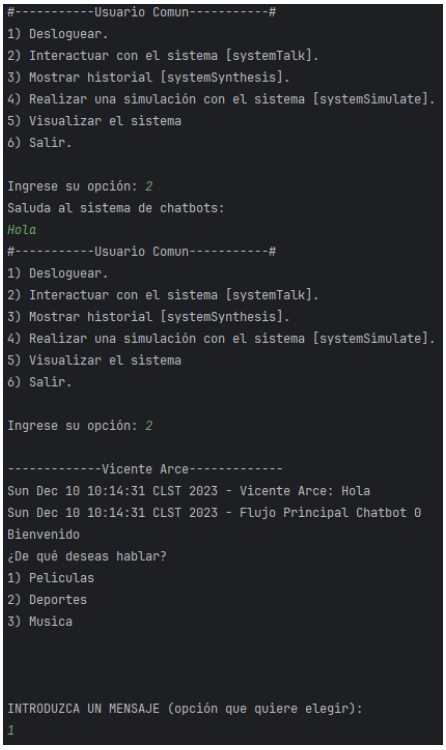


Figura 4: Hablando con el sistema.

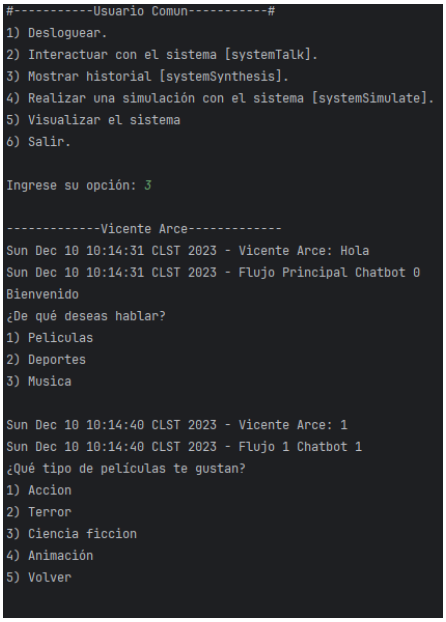


Figura 5: Mostrando historial.

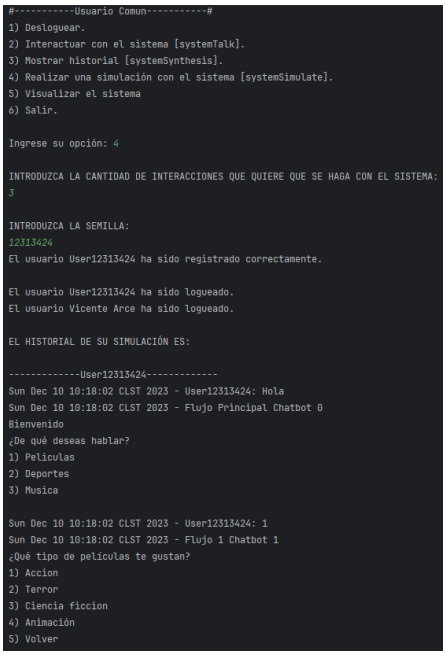


Figura 6: Haciendo una simulación del sistema.

Figura 7: Visualizando sistema.

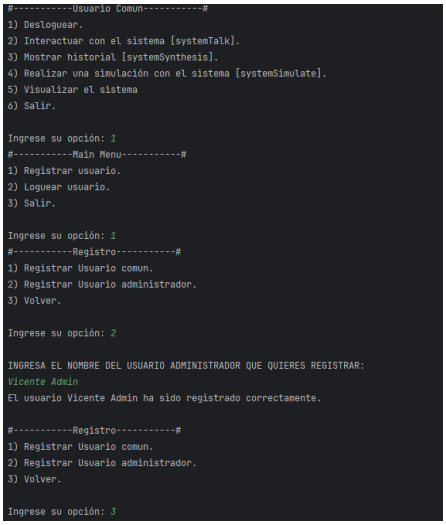


Figura 8: Deslogueando y registrando usuario administrador.

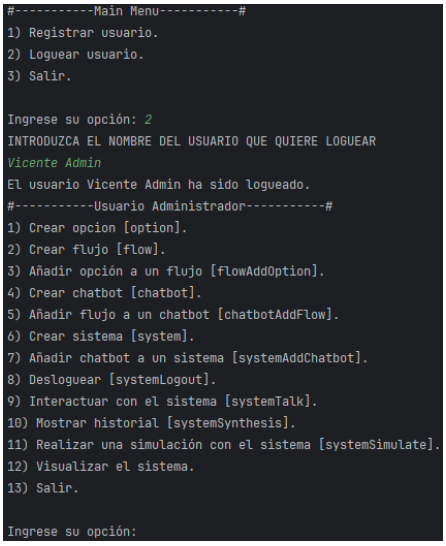
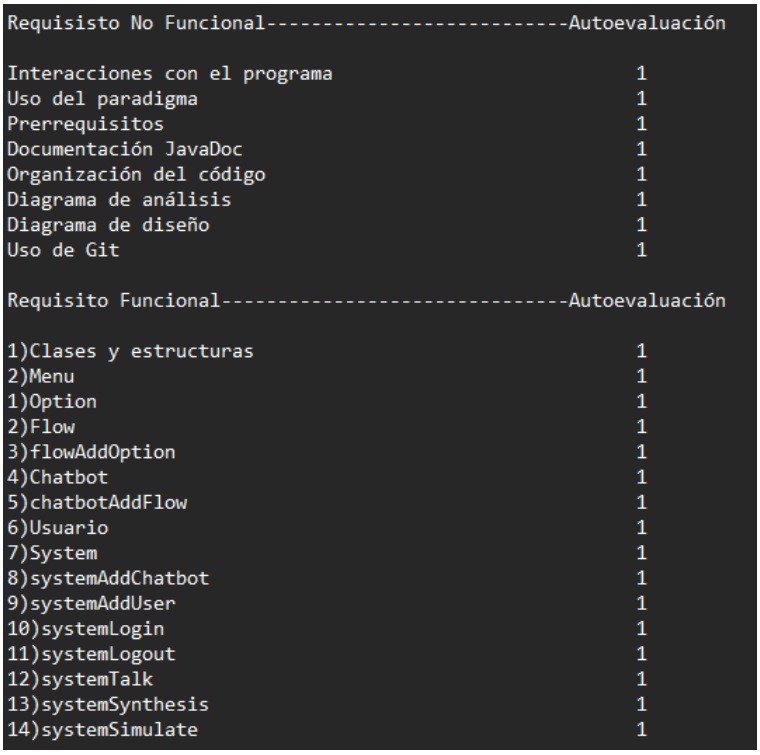


Figura 9: Logueando usuario administrador y opciones que trae

5.4. Anexo 4: Autoevaluación



Requisisto No Funcional	Autoevaluación
Interacciones con el programa	1
Uso del paradigma	1
Prerrequisitos	1
Documentación JavaDoc	1
Organización del código	1
Diagrama de análisis	1
Diagrama de diseño	1
Uso de Git	1

Requisito Funcional	Autoevaluación
1)Clases y estructuras	1
2)Menu	1
1)Option	1
2)Flow	1
3)flowAddOption	1
4)Chatbot	1
5)chatbotAddFlow	1
6)Usuario	1
7)System	1
8)systemAddChatbot	1
9)systemAddUser	1
10)systemLogin	1
11)systemLogout	1
12)systemTalk	1
13)systemSynthesis	1
14)systemSimulate	1

Figura 10: Autoevaluación Requisitos funcionales y no funcionales.