UNIVERSIDAD DE SANTIAGO DE CHILE

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Paradigmas de programación

Laboratorio: Paradigma Orientado a Objetos (Stackoverflow)

PROFESORES

Roberto González
Camila Marquez

INTEGRANTES

Javiera Tapia Bobadilla

SANTIAGO, CHILE

13 DE ENERO DE 2020

Índice

Tabla de contenidos

Introducción

Resolver un problema de programación requiere elegir los conceptos correctos. Todos, inclusive los más pequeños problemas requieren diferentes set de conceptos para abarcar sus distintas aristas, es por esto que los lenguajes de programación soportan varios paradigmas de programación. Un **paradigma de programación** es, en efecto, un *approach* a la programación de computadores basado en la teoría matemática, también puede ser descrito como un set de principios coherentes, un modelo o un marco de referencia. Cada paradigma soporta un set de conceptos que lo hacen más adecuado para un cierto tipo de problema, por ejemplo, la **programación orientada a objetos** es adecuada para problemas con un largo número de TDAs organizados jerárquicamente.

El **paradigma orientado a objetos** está basado en el concepto del objeto, que representa una abstracción de entidad de la realidad, tanto concreta como abstracta. Estos objetos pueden contener características (atributos) y comportamientos (métodos). Este paradigma es usado para estructurar un software en piezas simples y reusables de código (usualmente llamadas clases), que son usadas para crear instancias individuales de objetos.

En este laboratorio se presentará el diseño, análisis e implementación de un clon de la aplicación **Stackoverflow** --sitio de preguntas y respuestas para programadores profesionales y entusiastas de la programación--, haciendo uso del lenguaje de programación **Java** (perteneciente al paradigma orientado a objetos) y sus lineamientos, que permitirá la construcción pertinente de Tipos de Dato Abstracto (TDA), análisis y diseño orientado a objetos, diagramas UML y abstracción de entidades, entre otros conceptos claves del paradigma.

Descripción breve del problema

El stack tecnológico del foro Stackoverflow está, originalmente, basado en un lenguaje de programación orientado a objetos[1] (*server side*), siendo uno de los paradigmas dominantes en el desarrollo de software hasta el día de hoy dentro de la industria.

En esta ocasión, la problemática principal se centra en cubrir elementos claves como: usuarios registrados, preguntas, respuestas, etiquetas, recompensas, etc. y algunos de sus comportamientos como: login, register, ask, vote, reward, entre otros que provee la plataforma real, haciendo uso del lenguaje de programación Java. Es así, como dentro de la resolución iterativa e incremental de este proyecto, se hará énfasis en el diseño y análisis de la programación orientada a objetos, la definición adecuada en el modelamiento de las relaciones entre clases, entre otros.

Por otra parte, la implementación estará atada a requerimientos específicos como el uso correcto del paradigma orientado a objetos lo que implica realizar un diseño alrededor de 4 pilares básicos conocidos como los 4 pilares del diseño orientado a objetos: **abstracción, herencia, encapsulación y polimorfismo**. *(Ver anexo 1: Pilares básicos de la OOP)*.

1 Jeff Atwood, "What was Stackoverflow built with?", *The Overflow Blog*, https://stackoverflow.blog/2008/09/21/what-was-stack-overflow-built-with/

Descripción breve del paradigma

El **paradigma orientado a objetos** se caracteriza por usar objetos de distinto tipo (clase), entre cada tipo tipo pueden variar sus características (atributos) y comportamientos (métodos), éstos últimos pueden leer y escribir las características de estos atributos.

- ***TDA****:* Un Tipo de Dato Abstracto es un tipo (o clase) de objetos donde su comportamiento está definido por un set de valores y operaciones.

- *Objeto:* Representa una abstracción de entidades de la realidad (sean concretas o abstractas)**.**

- *Abstracción:* Manejar complejidad escondiendo detalles innecesarios para el usuario. Esto permite que el usuario implemente lógica más compleja sobre la abstracción provista sin necesidad de entender o incluso conocer los detalles internos de la implementación.

- *Herencia:* Los objetos se pueden crear como subtipos de otros objetos, heredando variables y métodos de estos objetos.

- *Encapsulación:* los objetos contienen data y métodos que afectan a esta última, lo que significa que una buena OOP provee métodos *getters* y *setters* que controlan el acceso a esta data. (Encapsulación).

- *Polimorfismo:* Literalmente, cambiar de forma. Este concepto permite que un objeto o método genérico, interface o un objeto regular sirva como una plantilla o template para otros objetos y métodos.

Análisis del problema (requisitos específicos)

El laboratorio requiere de la implementación de diversas funciones obligatorias que cumplen con la simulación del software *Stackoverflow* bajo el alero del paradigma orientado a objetos. El diseño de abstracciones y las características de éstas son fundamentales en primera instancia, puesto que los objetos requeridos deben combinarse de forma apropiada (bajo acoplamiento, alta cohesión) y actuar de forma consecuente al diseño planteado *(Ver anexo 2: Diagrama de análisis)*, el que será mejorado y refactorizado revisando los requerimientos de este proyecto.

Requerimientos funcionales

1) **Menú interactivo:** El programa debe incluir un menú por terminal/consola que permita la interacción del usuario con la solución, implementando las entidades y funcionalidades que permitan utilizar su simulación del sistema de preguntas y respuestas. Las acciones deben dar feedback si pudieron o no concretarse con éxito. Se maneja sólo para inicializar la aplicación, sin embargo, no se considera parte del modelado de la aplicación.

2) **Register*:*** Funcionalidad que permite el registro de nuevos usuarios en la plataforma, además de su ingreso y salida a la plataforma con sus credenciales. Se ejecuta por medio de una implementación de una *Interface* (Auth) en el stack. Este registro de usuario es único por lo que se valida su unicidad.

3) **Login/logout**: Funcionalidad que permite autenticar a un usuario previamente registrado (dentro del stack), para lo anterior, se verifica la existencia del usuario registrado. Si la autenticación es válida (username y password son correctos), el retorno es *true*. El usuario autenticado se asocia a una instancia de stack.

4) **Ask:** Funcionalidad que permite añadir una respuesta a una pregunta dentro del Stack, dejándola almacenada en la estructura correspondiente (lista de preguntas de Stack). Este método modificará la instancia de Stack, acción que incluirá esta pregunta en una lista de preguntas dentro del objeto Stack, validando que el usuario se encuentra logueado inicialmente.

5) **Answer:** Funcionalidad que permite consultar el valor que toma un Stack al realizar la respuesta a una pregunta. Esta funcionalidad debe mostrar una lista con todas las preguntas del Stack, permitiendo elegir una de las preguntas para poder escribir una respuesta a ésta. Sólo los usuarios con sesión iniciada pueden responder preguntas. (Auth)

6) **Accept:** Funcionalidad que permite a un usuario (con sesión iniciada en la plataforma) aceptar una respuesta a una de sus pregunta, acción que debe verse reflejada a través del menú interactivo. Si hay recompensas activas en la pregunta, estas deben resolverse otorgando los puntos de reputación al usuario. Sólo los usuarios con sesión iniciada pueden aceptar respuestas.

4) **Reward:** Funcionalidad que permite a un usuario (con sesión iniciada en la plataforma) ofrecer una recompensa para una determinada pregunta y calcular recompensas y reputaciones asociadas. Una vez que una respuesta es aceptada, la recompensa se descuenta definitivamente de la reputación del autor y se traspasa al usuario que entregó la respuesta.

5) **Vote:** Funcionalidad que permite a un usuario votar por una pregunta o respuesta. El voto puede ser positivo o negativo, ajustando los puntajes correspondientes.

Diseño de la solución

Respecto al diseño de la solución planteada para este laboratorio, se procura cumplir con los principios y patrones sugeridos para este paradigma y cumplimiento de los requerimientos funcionales de este laboratorio, algunos de ellos:

- Iterar sobre el problema asignado, analizando y diseñando una solución, resolviendo los **qué** y **cómo**.
- Diagramar y analizar las entidades involucradas, características, comportamientos y la relaciones entre ellas, eliminando ambigüedades.
- Procurar que el diseño de clases no viole los principios de **bajo acoplamiento y alta cohesión.** Se cuida que el nivel de interdependencia entre módulos sea bajo y que la especificidad de las entidades sea alto.
- Hacer uso de los 4 pilares básicos de la OOP (si aplica): abstracción, herencia, encapsulación y polimorfismo.

De lo anterior, luego de hacer un diagrama de análisis inicial de la problemática, se plantea una solución que involucra las siguientes entidades principales:

- **Stack**
- **Question**
- **Answer**
- **Reward**
- **Label**
- **User**

Luego, cada requerimiento funcional busca que las entidades manejen su propia responsabilidad dentro del sistema a través de las clases (entidades) anteriormente mencionadas. La idea principal del desarrollo de este laboratorio fue el apropiado uso de patrones y abstracciones coherentes a lo solicitado, proporcionando un marco definido, encapsulado y adecuado para los requerimientos del cliente (en este caso, a través de un enunciado).

Idear y conceptualizar los requerimientos *a priori*, siguiendo un orden claro nos facilita la implementación de código, sin lugar a dudas. Antes de programar, es necesario analizar y aplicar una arquitectura correcta para evitar refactorizaciones innecesarias, documentar nuestro código, entre otras prácticas de software.

Finalmente, se definen algunas relaciones (*Ver Anexo 3: Diagrama de diseño*) entre entidades como sigue (sin duplicar):

- La clase Stack **compone** a las clases *User, Question* y *Label*. Se quiere que las instancias de estas últimas clases dependan de la existencia de un Stack ya que este representa un contenedor de información.
- La clase User se **asocia** (dependencia débil) a las clases *Question*, *Answer* y *Reward* permitiendo así asociar al autor de la creación de estas instancias a un usuario en particular.
- La clase Question **agrega** (tiene) a las clases Answer, Reward y Labels. Se busca que el tiempo de vida de los objetos incluidos sea independiente de Question (no se inicializan como parte de Question).

Aspectos de implementación

Estructura del proyecto

La estructura del proyecto se basa en la modularización de cada TDA elegido, para ello, cada archivo .java maneja su propia lógica, siguiendo los principios de encapsulación. Luego, se maneja un único paquete o unidad lógica, denominado **project**, directorio que alberga todas las entidades definidas en el diagrama de diseño de clases (UML), agregando una entidad extra que facilita el menú (**Menu.java**) usado por el usuario a través del terminal. Este menú podrá ser compilado a través de la herramienta **Gradle** que será detallado en las instrucciones de uso, ya que está diseñada para ser flexible para compilar casi todo tipo de software.

Bibliotecas empleadas

Para las funcionalidades y, de acuerdo a los especificado en las instrucciones de este laboratorio, se hace uso exclusivamente de la biblioteca estándar de java, sin tener dependencias externas, como las disponibles en formato JAR.

Detalles del compilador, lenguaje e intérprete

- **Lenguage**: Java (https://docs.oracle.com/en/java/), OpenJDK versión 11
- **Intérprete**: IntelliJ IDEA Community 2020.3
- **Compilado**r: Gradle
- **Sistema operativo utilizado**: OSX

Instrucciones de uso

Instrucciones generales

- Descargar repositorio especificado en el archivo repositorio.txt adjuntado en el directorio zippeado (.zip) subido a la plataforma uvirtual.
- Descargar la herramienta **Gradle** e instalarla de acuerdo a la guia: https://docs.gradle.org/current/userguide/getting_started.html
- Ejecutar comando en el root del proyecto para correr compilador Gradle en terminal: **`./gradlew run`**

Ejemplos

La aplicación debe permitir cargar de forma inicial al menos 1 Stack, 4 usuarios registrados, 5 preguntas, 10 respuestas y 3 etiquetas. De esta forma, el menú principal permite crear un Stack o bien comenzar con uno cargado en el sistema. El Stack se imprime en cada acción realizada en el programa para una mejor visualización de las modificaciones. *(Ver Anexo 4: Resultado de pruebas)*.

Resultados esperados

Se espera que cada uno de los ejemplos cumpla las expectativas mencionadas en los requerimientos funcionales, exponiendo una estructura coherente y clara.

Posibles errores

No se reconocen posibles errores, puesto que los ejemplos se ajustan a los requerimientos. Se recomienda no saturar de casos bordes el menú principal, ya que no se implementan **try catch** suficientes para opciones inválidas en todos los caso

Resultados y autoevaluación

Requerimientos	Grado de alcance (%)	Pruebas realizadas	% de pruebas exitosas	% de pruebas fracasadas[2]
register	100%	(3) Casos de registros exitosos. (1) Caso particular (validación de existencia)	100%	0%
login/logout	100%	(3) Casos de login exitosos y (1) caso particular (validación de usuario registrado)	100%	0%
ask	100%	(3) Casos de creación de preguntas exitosas y (1) caso particular (creación de preguntas múltiples).	100%	0%
answer	100%	(2) Casos de creación de respuestas exitosas y (1) caso particular (creación de preguntas múltiples) y (1) caso que valida existencia de pregunta.	100%	0%
reward	100%	(2) Casos de aceptación de respuestas caso particular y (1) Caso que valida la aceptación de una respuesta perteneciente al usuario.	100%	0%
accept	100%	(2) Casos válidos de aceptación. y (1) Caso de pregunta con respuesta ya aceptada.	100%	0%
vote	100%	(4) casos de votación en contra y favor tanto para preguntas como para respuestas.	100%	0%

En la tabla anterior, no se presentan métodos que no se completaron dado que en el laboratorio se trabajaron todos los requerimientos obligatorios y **uno** de aquellos opcionales (vote) en su totalidad.

[2] Durante el desarrollo del laboratorio, muchas pruebas fueron un fracaso, sin embargo, se trabajó en reducir al máximo los bugs y efectos secundarios hasta lograr los resultados esperados.

Conclusiones del trabajo

Este trabajo ha permitido conocer en detalle el alcance del Paradigma Orientado a Objetos (POO), el que permite modelar y organizar un software diseñado alrededor de una data u objetos, en vez de funciones o lógica. Un objeto puede ser definido como un campo de data que tiene atributos y comportamientos únicos, permitiendo así que cualquier persona tenga la posibilidad de comprender fácilmente, un código de estas características.

Cabe mencionar que también hay algunas ventajas del paradigma, en las que se pueden mencionar: **reusabilidad, mantenibilidad, modificabilidad y fiabilidad**, características que no por nada hasta el día de hoy hacen de este paradigma uno de los más usados y respetados por la comunidad informática.

Sin embargo, y, como todo, hay una desventaja, por ejemplo, el cambio en la forma de pensar de la forma tradicional a la orientada a objetos, la ejecución de programas orientados a objetos puede ser más lenta y la necesidad de utilizar bibliotecas de clases que obligan a su aprendizaje y entrenamiento, entre otras cosas. Asimismo, cabe resaltar que en el paradigma hay pilares básicos como la **abstracción, herencia, polimorfismo y encapsulación** siendo éstas elementales para una implementación acertada de una idea o diseño.

Respecto a las dificultades, se puede mencionar que, personalmente, se considera al lenguaje de programación Java como uno de los más verbosos del paradigma, aún así, permite una representación clara de las principales características de éste.

Referencias

Atwood, Jeff. (21 de septiembre de 2008). *What was Stackoverflow built with?*. The Overflow Blog. https://stackoverflow.blog/2008/09/21/what-was-stack-overflow-built-with/

Amarasinghe, Saman., Chlipala, Adam., Devadas, Srini., Ernst, Michael., Goldman, Max., Guttag, John., Jackson, Daniel., Miller, Rob., Solar-Lezama, Armando. (s.f.). *Abstract Data Types*. MIT. http://web.mit.edu/6.005/www/fa14/classes/08-abstract-data-types/

Algunos aspectos de este trabajo fueron vistos de manera general en (https://stackoverflow.blog)

Documentación Java extraída desde (https://docs.oracle.com/en/java/)

Ehud Sterling Leon; Shapiro . (Enero de 1994). *The Art of Prolog*.

Van Roy, Peter. (Abril de 2012). *Programming Paradigms for Dummies: What Every Programmer Should Know*. Researchgate. (https://www.researchgate.net/publication/241111987_Programming_Paradigms_for_Dummies_What_Every_Programmer_Should_Know)

Anexos

Anexo 1: Cuatro pilares fundamentales de OOP

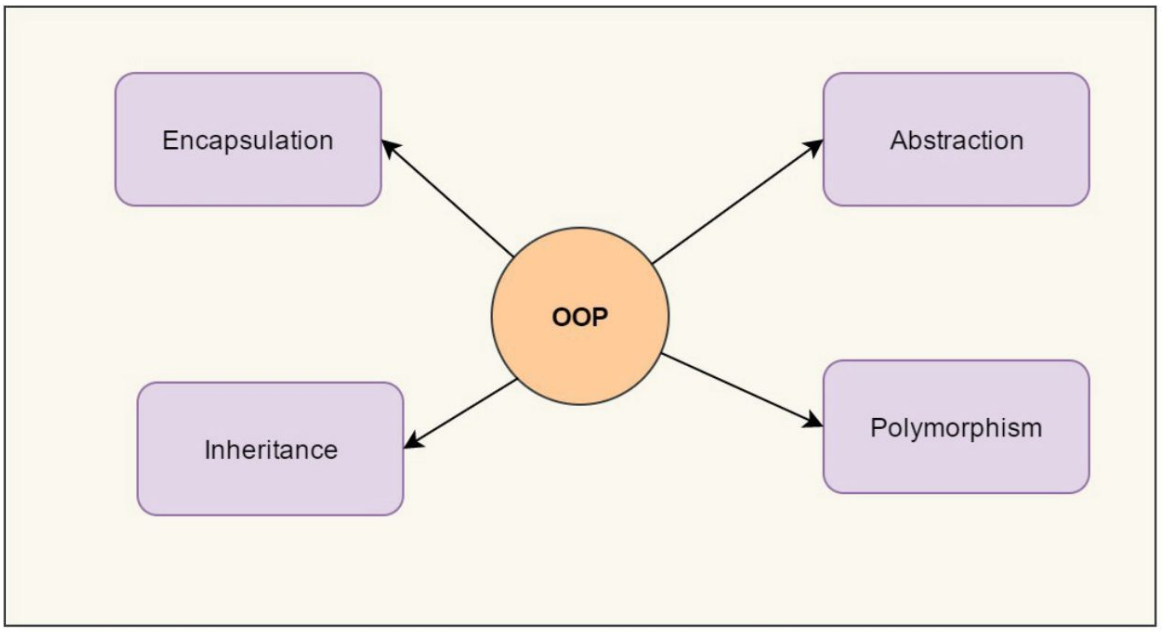

Figura: Cuatro pilares de la programación orientada a objetos

Anexo 2: Diagrama de análisis

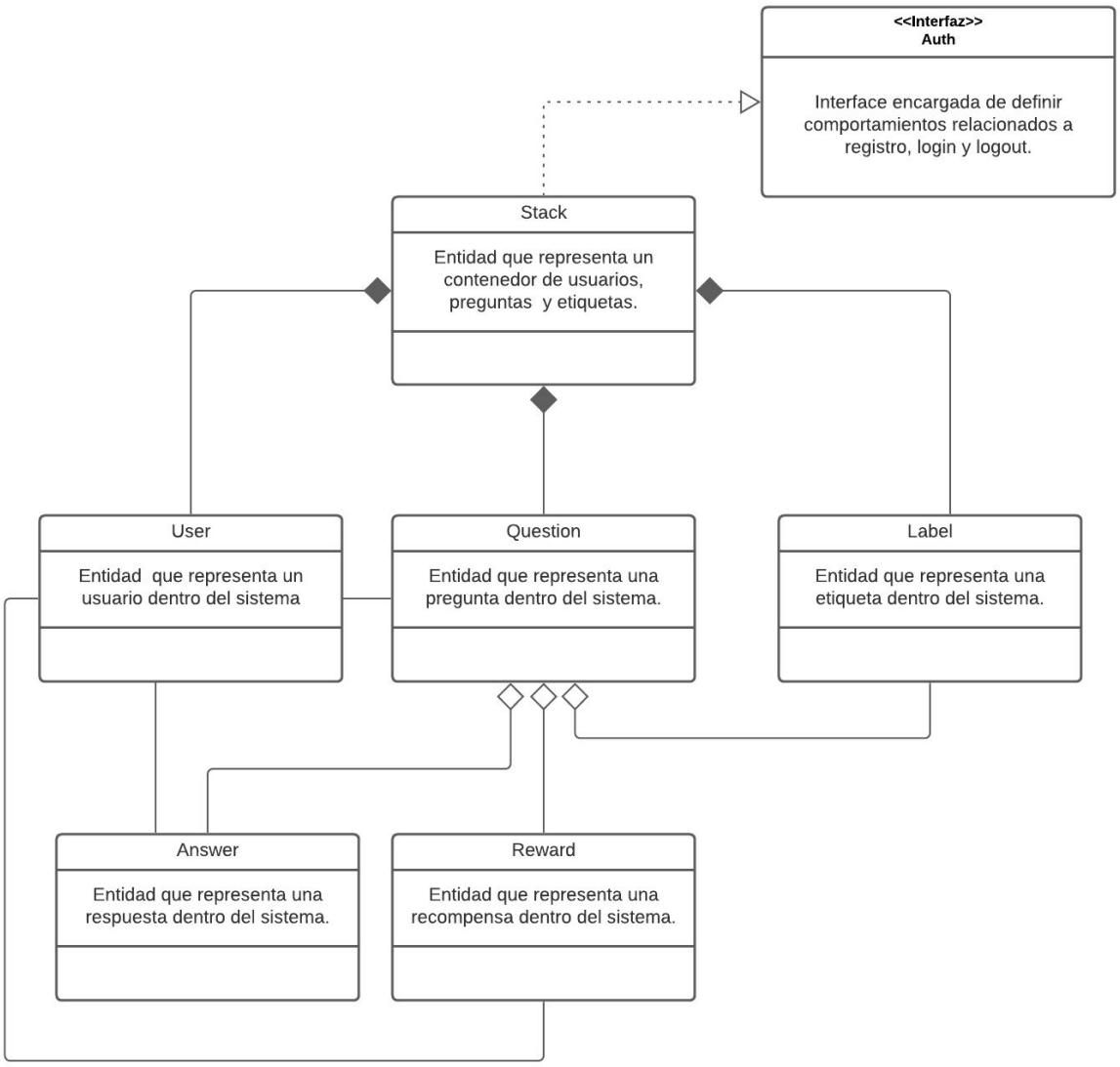

Figura: Diagrama de análisis inicial (borrador)

Anexo 3: Diagrama de diseño

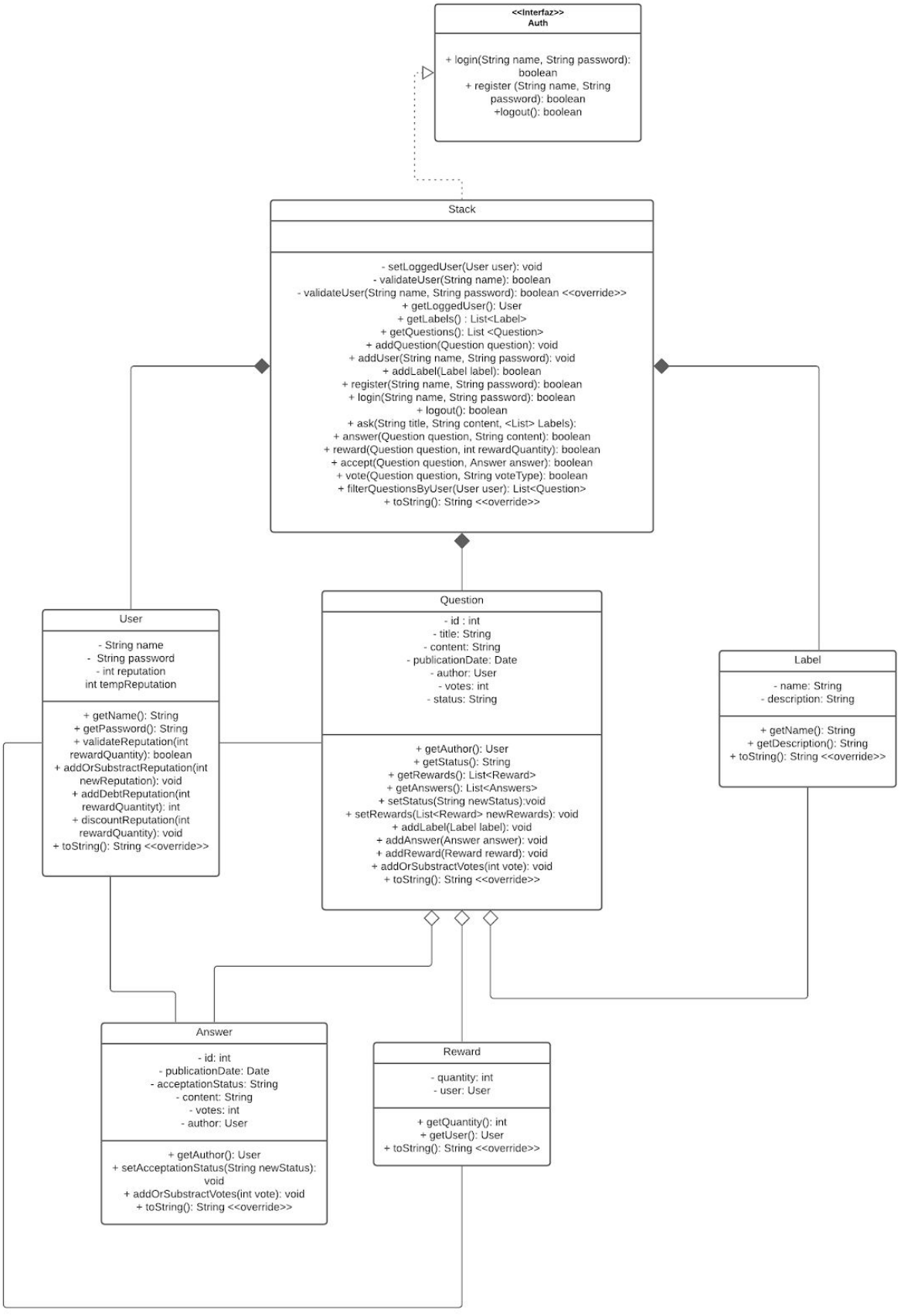

Figura: Representa Diagrama de clases UML final

Anexo 4: Resultado de pruebas

Figura: Impresión de Stack en cada acción del programa, facilitando la visualización de la estructura.