Laboratorio N°1 Paradigmas de la Programación:

SISTEMA DE LOGROS

# Alba Francisco, Nahman Martina, Verón Facundo

Lic. Cs. de la Computación/Facultad de Ingeniería/UNCuyo 20 de septiembre de 2023

**Abstract**

En el presente documento procederemos a explicar el desarrollo del ejercicio integrador correspondiente al Laboratorio N°1 de la materia Paradigmas de Programación. La temática del mismo trata de la correcta implementación de un Sistema de Logros para juegos.

# 1. Introducción

En el presente informe buscamos encontrar una forma óptima de implementar un sistema de logros que permita hacer seguimiento del cumplimiento de determinados objetivos aplicados a un juego dado. Para ello, hacemos uso de los conocimientos adquiridos a lo largo del cursado de la cátedra, que involucran aspectos fundamentales de la programación orientada a objetos, tales como abstracción, herencia, encapsulamiento y polimorfismo.

# Metodología de trabajo

Luego de haber realizado una primera lectura del enunciado propuesto, debatimos sobre lo comprendido respecto a los requisitos de la implementación y del respectivo modelo, llegando a la conclusión de que debíamos indagar más acerca del tema, y consultar nuestras dudas con el profesor. Posteriormente, procedimos a la construcción de un primer boceto de modelo, que incluyó nombres de las clases y sus atributos correspondientes, y ya habiendo podido despejar algunas de nuestras dudas relacionadas a dicho modelo, incorporamos también algunos métodos o servicios que consideramos relevantes para la funcionalidad de cada clase modelada.

## 2.1 Diseño

## Comenzamos así con el trabajo de implementar en código del modelo obtenido, para lo cual empleamos la versión 11 del JDK de Java y los entornos de desarrollo IntelliJ Idea y Eclipse.

## 2.1.1 Primera iteración

## Nuestra primera idea de solución propuso la creación de distintas responsabilidades en la clase “User” que se correspondieron con posibles acciones de la vida real, tales como correr, caminar, jugar un cierto tiempo o mirar un número determinado de introducciones. La realización de esas acciones derivaría entonces en la adquisión de determinadas insignias; lo cual a su vez implica la obtención de un puntaje adicional correspondiente que viene definido por la propia insignia. La utilización sucesiva del sistema por parte del usuario, le permitiría así ir acumulando puntaje y subir de membresía para poder acceder a productos exclusivos dentro de la tienda. El modelo correspondiente al sistema descripto puede observarse en la Figura 1.

## 2.1.2 Problemas encontrados y soluciones intermedias

## Al intentar implementar en código la solución mencionada anteriormente, nos dimos cuenta de que las clases “User” y “AchievementSystem” acumulaban cada vez más y más responsabilidades, por lo que nos alejábamos entonces del objetivo de lograr una alta cohesión y un bajo acoplamiento. Nos detuvimos entonces a analizar el problema, e intentamos resolverlo proponiendo la creación de una clase abstracta “Action” que nos permitiese modelar un tipo genérico de acción a realizar por el usuario y crear a partir de ese molde las distintas acciones con su comportamiento específico haciendo uso del concepto de herencia. El diagrama de la mencionada alternativa está representado en la Figura 2.

## Llegado este punto, y luego de realizar distintas pruebas y analizar un poco más en detalle el problema al momento de programar, observamos que seguíamos asignando mal ciertas responsabilidades (como las de realizar una acción particular o ganar una insignia) y decidimos volcar nuestras inquietudes con el profesor. Confirmamos así nuestra sospecha y llegamos a la conclusión de que estábamos perdiendo de vista el objetivo principal del ejercicio, que es desarrollar un Sistema de Logros y no necesariamente el juego ni el conjunto de acciones provistas por el mismo. Así que eliminamos de esta forma el modelado de acciones e incorporamos nuevas clases (tales como “Account”, “GreatRunner”, “Experienced” y “Globetrotter”) y enumeraciones (tales como “PaymentMethod”, “Membership” y “ProductType”) al desarrollo de la solución, lo cual nos permitió esta vez avanzar en forma más coherente con el objetivo principal y con las pautas de la programación orientada a objetos que aprendimos en clase. El diagrama que resultó de la implementación de estos cambios puede visualizarse en la Figura 3.

## 

**Figura 1: Primera iteración del modelo en lenguaje UML**

## 

**Figura 2: Posible modelado de las acciones de un juego**

## 

**Figura 3: Penúltima aproximación al diagrama UML solución del problema**

## 2.1.3. Versión final del modelo y resumen de las decisiones tomadas

## 

## Figura 4: Aproximación final al diagrama UML solución del problema

## La Figura 4 en la parte superior representa el diagrama final de nuestra solución al problema inicial, luego de haber iterado una vez más sobre el diagrama anterior y depurado algunos detalles e inconsistencias que fuimos encontrando. Entre esos detalles podemos mencionar:

## El replanteo de la forma en que creamos y funcionalizamos las insignias, basándonos en el hecho de que previamente, en la situación correspondiente al diagrama de la Figura 3, las insignias derivadas de la clase “Badge” no aportaban algún atributo o comportamiento especial que justificara tener que evitar instanciar directamente a la clase “Badge”. Fue así que optamos por hacer que la clase “Badge” deje de ser abstracta, eliminar también las clases “GreatRunner” y “Experienced”, e incorporar las clases “Golden” y “Master”. De esta manera, ahora instanciando alguna de las clases “Globetrotter”, “Golden” ó “Master” podemos crear medallas que otorguen al usuario una cantidad especial de puntos o una recompensa particular de dinero. Y, al mismo tiempo, si queremos crear un tipo de insignia genérico, tenemos la libertad de hacerlo directamente creando un nuevo objeto de la clase “Badge”. Siguiendo la mecánica descripta, incorporamos entonces al sistema un total de 14 insignias disponibles para repartir entre los usuarios.

## La separación del comportamiento de cada entidad en clases “service”. Esto no fue estrictamente requerido por alguna consigna o funcionalidad que necesitemos, pero sí nos permitió organizar mejor la lógica del código y mejorar la posibilidad de realizar modificaciones de cara al futuro.

## La incorporación de una clase que provea un método encargado de procesar la información ingresada por el usuario en la consola. Designamos a la misma con el nombre “StringUtil”, y nos resultó importante su implementación debido al uso muy frecuente de la clase “Scanner” de Java en distintas partes del código cada vez que nos topábamos con la necesidad de soliciar un dato al usuario.

## 2.2 Planificación

## La forma más efectiva que encontramos para llevar a cabo el proyecto fue juntarnos a debatir ideas y propuestas acerca de este. Respecto al código, fueron eficientes las reuniones presenciales y virtuales para poder avanzar en conjunto. Además, nos repartimos algunas implementaciones para poder avanzar de forma asíncrona.

## 2.3 Implementación

## (en construcción)

## 2.4 Instrucciones para el usuario

# 3. Conclusión