Universidad Nacional de La Matanza

PARADIGMAS DE PROGRAMACIÓN

Trabajo práctico N°2

Héroes y Villanos

Integrantes:

- Arias, Gabriel
- Céspedes, Cristian
- Collazo, Ignacio
- Nogueira, Ignacio
- Villa, Gabriel

Índice

Trabajo Práctico

Introducción

El presente trabajo tiene como objetivo afianzar la práctica de Programación Orientada a Objetos. En particular, de los mecanismos de herencia, polimorfismo, clases abstractas, interfaces, entrada/salida y patrones de diseño.

Se desea modelar un juego compuesto por héroes y villanos. El mecanismo básico del juego se basa en enfrentar un personaje con otro y decidir cuál de ellos es el ganador.

Puntos para tener en cuenta:

- Cada personaje del juego posee un nombre real, un nombre de superhéroe o villano y un conjunto de características. Las características son: velocidad, fuerza, resistencia y destreza.
- Para decidir quién es el ganador se utiliza el valor de una de las características. En caso de empate, se decide por el valor de otra característica dada (en el orden establecido, y volviendo a comenzar si fuera necesario).
- El juego debe proveer un mecanismo de agrupamiento de los personajes en ligas para realizar desafíos o enfrentamientos entre grupos de personajes. Para esto, el valor grupal de cada característica se determina como el promedio de los valores de esa característica entre todos los personajes.
- Las ligas se almacenan en un archivo de texto, y se cargan en memoria antes de un enfrentamiento.
- Las Ligas son homogéneas, sólo de Héroes o sólo Villanos. Cada personaje solamente puede pertenecer a una liga (o a ninguna).
- Todo se realizará por medio de la consola, no se esperan interfaces gráficas.
- Solo se enfrentan Héroes contra Villanos (o ligas de Héroes contra Ligas de Villanos).

Decisiones de Diseño

Respecto a este punto, se listarán las decisiones de diseño principales en torno a los componentes fundamentales del sistema:

- Personajes y Ligas
    - Se optó por utilizar una clase abstracta UnidadCompetidor (quien gestionará los elementos en común de los personajes y ligas), la cual es heredada por la clase Competidor y Ligas.
    - Cada unidad (personaje o liga) contendrá su paquete de características, a través de la construcción de un objeto de la clase

Característica, diseñada para tal fin.

- La creación de unidades (personajes o ligas), puede realizarse a través de la carga de sus correspondientes archivos, o a través de la creación manual, ingresando los valores (que son validados) necesarios para crear una unidad.
- Las ligas pueden ser integradas por una o más ligas y, además, uno o más personajes.
- Como patrón de diseño se optó por implementar Composite, el cual fue de utilidad para la creación de ligas más grandes, como el caso de ligas integradas por otras ligas y/o personajes.

- Combates
  - Los combates se realizarán entre:
    - Personaje vs Personaje
    - Personaje vs Liga
    - Liga vs Liga
  - Los combates se realizarán a través de un método enfrentar(), que tomará por parámetro una característica especificada por el usuario, y realizará el enfrentamiento entre dos unidades, lo que abarca cualquier caso: personaje vs personaje, personaje vs liga y liga vs liga.
- Reportes
  - Los reportes se generan automáticamente luego de la selección de liga/personaje del usuario, evitando el ingreso manual de datos que podrían propiciar fallas.

Archivos utilizados

- **Paquete Archivos**
  - *ArchivoLigas.java* → Utilizado para gestionar la carga de los personajes, ingresados a través del archivo "personajes.in".
  - *ArchivoPersonajes.java* → Utilizado para gestionar la carga de las ligas, ingresadas a través del archivo "ligas.in".
- **Paquete Comparadores**
  - *ComparadorPorBando.java* → Utilizado para realizar la comparación de los personajes competidores, y ordenarlos dado un bando específico.
  - *ComparadorPorCaracteristica.java* → Utilizado para realizar la comparación de las características de los personajes, y ordenarlos dado un bando específico.
- **Paquete Excepciones**
  - *CaracteristicaExcepcion.java* → Utilizado para el manejo de

excepciones elevadas por la clase Característica.

- *CompetidorExcepcion.java* → Utilizado para el manejo de excepciones elevadas por la clase Competidor.
- *SistemaExcepcion.java* → Utilizado para el manejo de excepciones elevadas por la clase SistemaHeroesVillanos.

- **Paquete Sistema**
  - *App.java* → Utilizado para ejecutar el menú principal del sistema. Es el punto de entrada de la ejecución.
  - *Bandos.java* → Enum utilizado para la validación de los bandos correspondientes a los personajes y ligas.
  - *Característica.java* → Utilizado para gestionar y empaquetar la serie de características que poseen tanto los personajes como las ligas (velocidad, fuerza, resistencia y destreza).
  - *Competidor.java* → Utilizado para la creación y gestión de los personajes, y la implementación de cómo obtener sus elementos.
  - *Liga.java* → Utilizado para la creación y gestión de las ligas, y la implementación de cómo obtener sus elementos.
  - *Menu.java* → Utilizado para la gestión interactiva entre el usuario y la consola, permitiendo usar los servicios esenciales del sistema, como administración de personajes, ligas, realización de combates y generación de reportes.
  - *SistemaHeroesVillanos.java* → Utilizado como pieza central del mecanismo del juego. Es el núcleo de la lógica del juego la cual incluye las funcionalidades principales, como administrar y crear personajes, administrar y crear ligas, realizar combates entre, generar los reportes. También se encarga del almacenamiento de los personajes y ligas en los archivos de output correspondientes.
  - *UnidadCompetidor.java* → Utilizado como clase abstracta contenedora de los elementos en común entre las clases Competidor y Liga, las cuales realizan sus implementaciones específicas dentro del ámbito correspondiente.
- **Paquete Tests**
  - *ArchivoLigasTests.java* → Utilizado para las pruebas unitarias referentes a la gestión del archivo “Ligas.in”.
  - *ArchivoPersonajesTests.java* → Utilizado para las pruebas unitarias referentes a la gestión del archivo “Personajes.in”.
  - *LigaTests.java* → Utilizado para las pruebas unitarias referentes a la gestión de la clase Liga.
  - *PersonajeTests.java* → Utilizado para las pruebas unitarias referentes a la gestión de la clase Personaje.
  - *SistemaHeroesVillanosTests.java* → Utilizado para las pruebas unitarias referentes a la gestión de la clase SistemaHeroesVillanos.

- *UnidadCompetidorTests.java* → Utilizado para las pruebas unitarias referentes a la gestión de la clase UnidadCompetidorTests.

Organización del trabajo

La organización del trabajo se basó en la asignación de los componentes más importantes del sistema a los integrantes. Para esto, tuvimos en cuenta las siguientes secciones:

Sección	Integrantes
Sistema central Héroes y Villanos	Arias, Gabriel Collazo, Ignacio Nogueira, Ignacio
Administración de personajes	Arias, Gabriel Collazo, Ignacio Nogueira, Ignacio
Administración de ligas	Arias, Gabriel Collazo, Ignacio Nogueira, Ignacio
Administración de archivos	Collazo, Ignacio Nogueira, Ignacio
Administración de reportes	Arias, Gabriel Nogueira, Ignacio
Administración de tests	Céspedes, Cristian Villa, Gabriel
Integración de patrones de diseño	Arias, Gabriel Collazo, Ignacio Nogueira, Ignacio
Manejo de excepciones	Céspedes, Cristian Nogueira, Ignacio
Gestión interactiva (Menú)	Arias, Gabriel Céspedes, Cristian Collazo, Ignacio
Modelado diagrama UML	Céspedes, Cristian Villa, Gabriel
Formateo y depuración de código	Céspedes, Cristian Villa, Gabriel
Generación del informe	Céspedes, Cristian Collazo, Ignacio Villa, Gabriel

Conclusiones

El desarrollo de este trabajo práctico propuso la utilización de los pilares fundamentales de la Programación Orientada a Objetos (POO), y ha permitido explorar la aplicación de patrones de diseño en el contexto de un juego interactivo de héroes y villanos.

A lo largo del trabajo práctico, hemos abordado los siguientes aspectos clave:

- **Herencia y Polimorfismo**: para modelar relaciones entre clases, permitiendo la reutilización de código y la creación de jerarquías estructuradas, además del tratamiento uniforme de objetos a través de interfaces y clases abstractas.
- **Encapsulamiento**: utilizamos el encapsulamiento para ocultar la implementación interna de las clases, promoviendo la modularidad y la mantenibilidad del código.
- **Abstracción**: La abstracción nos permitió modelar conceptos esenciales y crear clases abstractas que encapsulan características comunes entre distintos objetos, fomentando la cohesión en el diseño.
- **Patrones de Diseño**: se aplicó el patrón de diseño Composite para resolver problemas recurrentes y mejorar la flexibilidad y extensibilidad del sistema.

La combinación de los pilares de la POO y los patrones de diseño ha resultado en un diseño eficiente y mantenible. La comprensión profunda de estos conceptos es crucial para desarrollar sistemas software sólidos y escalables.

La importancia de estos fundamentos en la creación de sistemas flexibles y fácilmente extensibles ha quedado claramente demostrada a lo largo del trabajo práctico.