Universidad Nacional de la Matanza

Trabajo práctico: Héroes y Villanos

Paradigmas de Programación.

Código de materia: 03646

Comisión: 02-2900

Docentes:

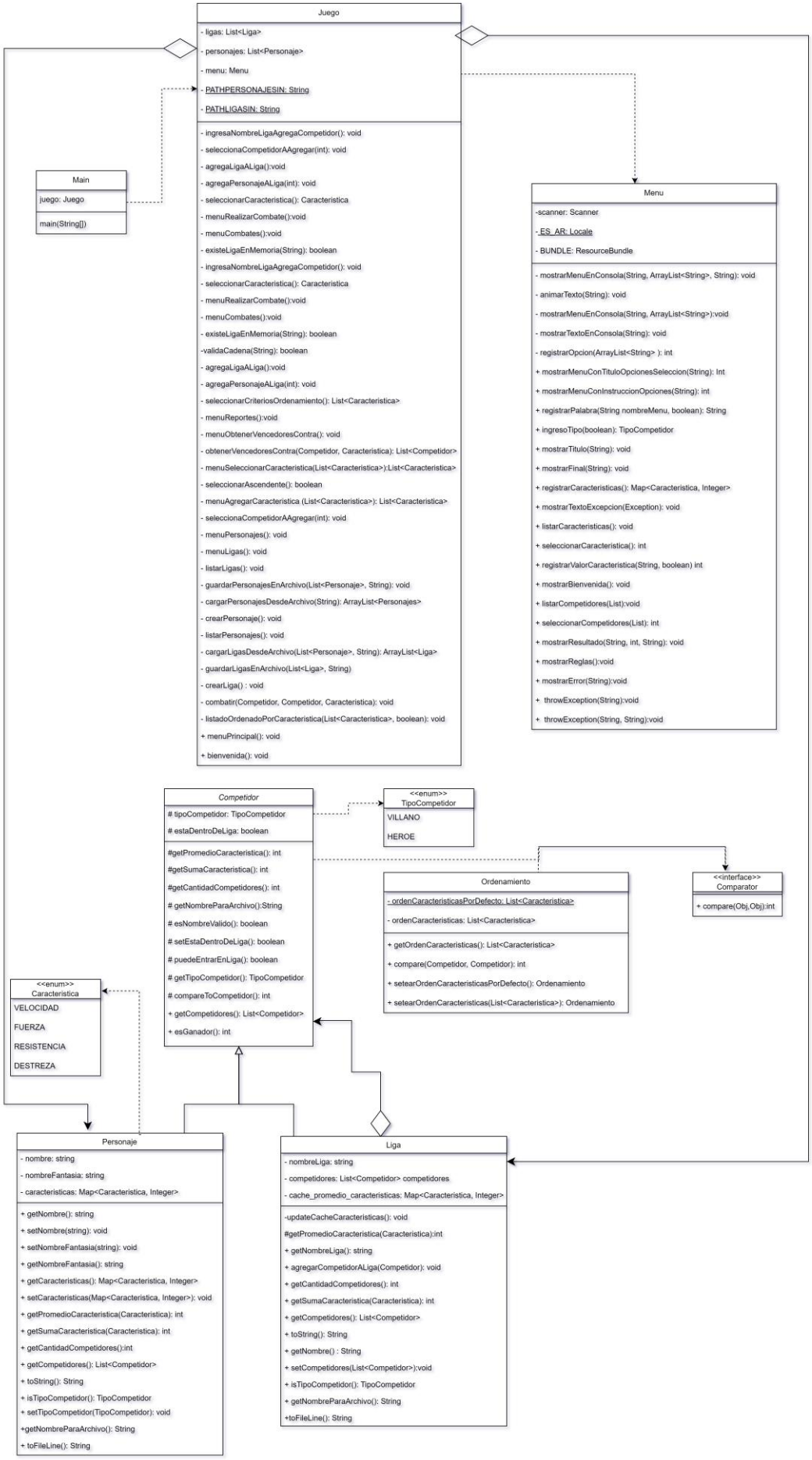
- Gasior, Federico Mauro.
- Lanzilotta, Hernán Gabriel.
- Videla, Lucas.

Integrantes:

- Arias, Tomás – 39.625.484
- Costa, Cristian Andres – 38.056.171
- La Giglia, Rodrigo – 33.334.248
- Lopez, Fernando – 42.537.132
- Ocampo, Pablo – 37.760.454

Link al código fuente:

https://github.com/cristian-costa/heroes-y-villanos


Juego
- ligas: List<Liga>
- personajes: List<Personaje>
- menu: Menu
- PATHPERSONAJESIN: String
- PATHLIGASIN: String
- ingresaNombreLigaAgregaCompetidor(): void
- seleccionaCompetidorAAgregar(int): void
- agregaLigaALiga():void
- agregaPersonajeALiga(int): void
- seleccionarCaracteristica(): Caracteristica
- menuRealizarCombate():void
- menuCombates():void
- existeLigaEnMemoria(String): boolean
- ingresaNombreLigaAgregaCompetidor(): void
- seleccionarCaracteristica(): Caracteristica
- menuRealizarCombate():void
- menuCombates():void
- existeLigaEnMemoria(String): boolean
-validaCadena(String): boolean
- agregaLigaALiga():void
- agregaPersonajeALiga(int): void
- seleccionarCriteriosOrdenamiento(): List<Caracteristica>
- menuReportes():void
- menuObtenerVencedoresContra(): void
- obtenerVencedoresContra(Competidor, Caracteristica): List<Competidor>
- menuSeleccionarCaracteristica(List<Caracteristica>):List<Caracteristica>
- seleccionarAscendente(): boolean
- menuAgregarCaracteristica (List<Caracteristica>): List<Caracteristica>
- seleccionaCompetidorAAgregar(int): void
- menuPersonajes(): void
- menuLigas(): void
- listarLigas(): void
- guardarPersonajesEnArchivo(List<Personaje>, String): void
- cargarPersonajesDesdeArchivo(String): ArrayList<Personajes>
- crearPersonaje(): void
- listarPersonajes(): void
- cargarLigasDesdeArchivo(List<Personaje>, String): ArrayList<Liga>
- guardarLigasEnArchivo(List<Liga>, String)
- crearLiga() : void
- combatir(Competidor, Competidor, Caracteristica): void
- listadoOrdenadoPorCaracteristica(List<Caracteristica>, boolean): void
+ menuPrincipal(): void
+ bienvenida(): void
Main
juego: Juego
main(String[])
Menu
-scanner: Scanner
- ES_AR: Locale
- BUNDLE: ResourceBundle
- mostrarMenuEnConsola(String, ArrayList<String>, String): void
- animarTexto(String): void
- mostrarMenuEnConsola(String, ArrayList<String>):void
- mostrarTextoEnConsola(String): void
- registrarOpcion(ArrayList<String>): int
+ mostrarMenuConTituloOpcionesSeleccion(String): Int
+ mostrarMenuConInstruccionOpciones(String): int
+ registrarPalabra(String nombreMenu, boolean): String
+ ingresoTipo(boolean): TipoCompetidor
+ mostrarTitulo(String): void
+ mostrarFinal(String): void
+ registrarCaracteristicas(): Map<Caracteristica, Integer>
+ mostrarTextoExcepcion(Exception): void
+ listarCaracteristicas(): void
+ seleccionarCaracteristica(): int
+ registrarValorCaracteristica(String, boolean) int
+ mostrarBienvenida(): void
+ listarCompetidores(List):void
+ seleccionarCompetidores(List): int
+ mostrarResultado(String, int, String): void
+ mostrarReglas():void
+ mostrarError(String):void
+ throwException(String):void
+ throwException(String, String):void
Competidor
tipoCompetidor: TipoCompetidor
estaDentroDeLiga: boolean
#getPromedioCaracteristica(): int
#getSumaCaracteristica(): int
#getCantidadCompetidores(): int
getNombreParaArchivo():String
esNombreValido(): boolean
setEstaDentroDeLiga(): boolean
puedeEntrarEnLiga(): boolean
getTipoCompetidor(): TipoCompetidor
compareToCompetidor(): int
+ getCompetidores(): List<Competidor>
+ esGanador(): int
<<enum>>
TipoCompetidor
VILLANO
HEROE
Ordenamiento
- ordenCaracteristicasPorDefecto: List<Caracteristica>
- ordenCaracteristicas: List<Caracteristica>
+ getOrdenCaracteristicas(): List<Caracteristica>
+ compare(Competidor, Competidor): int
+ setearOrdenCaracteristicasPorDefecto(): Ordenamiento
+ setearOrdenCaracteristicas(List<Caracteristica>): Ordenamiento
<<interface>>
Comparator
+ compare(Obj,Obj):int
<<enum>>
Caracteristica
VELOCIDAD
FUERZA
RESISTENCIA
DESTREZA
Personaje
- nombre: string
- nombreFantasia: string
- caracteristicas: Map<Caracteristica, Integer>
+ getNombre(): string
+ setNombre(string): void
+ setNombreFantasia(string): void
+ getNombreFantasia(): string
+ getCaracteristicas(): Map<Caracteristica, Integer>
+ setCaracteristicas(Map<Caracteristica, Integer>): void
+ getPromedioCaracteristica(Caracteristica): int
+ getSumaCaracteristica(Caracteristica): int
+ getCantidadCompetidores():int
+ getCompetidores(): List<Competidor>
+ toString(): String
+ isTipoCompetidor(): TipoCompetidor
+ setTipoCompetidor(TipoCompetidor): void
+getNombreParaArchivo(): String
+ toFileLine(): String
Liga
- nombreLiga: string
- competidores: List<Competidor> competidores
- cache_promedio_caracteristicas: Map<Caracteristica, Integer>
-updateCacheCaracteristicas(): void
#getPromedioCaracteristica(Caracteristica):int
+ getNombreLiga(): string
+ agregarCompetidorALiga(Competidor): void
+ getCantidadCompetidores(): int
+ getSumaCaracteristica(Caracteristica): int
+ getCompetidores(): List<Competidor>
+ toString(): String
+ getNombre() : String
+ setCompetidores(List<Competidor>):void
+ isTipoCompetidor(): TipoCompetidor
+ getNombreParaArchivo(): String
+toFileLine(): String

Decisiones de diseño.

1. Uso de patrones de diseño.

- Composite Pattern: En la implementación de la clase Liga, que puede contener Personaje y otras Liga, se sugiere el uso del patrón Composite para tratar grupos de competidores como un solo competidor.

- Strategy Pattern: La clase Ordenamiento utiliza el patrón Strategy al permitir cambiar dinámicamente el orden de comparación de las características, lo que refleja diferentes estrategias de comparación sin necesidad de modificar los objetos que están siendo comparados.

2. Modelo de dominio.

- Abstracción de Competidor: Se eligió una clase abstracta Competidor para definir la interfaz común y las operaciones para todas las entidades que pueden entrar en combate. Esto considera relevante la necesidad de tratar tanto a los personajes como a las ligas de manera uniforme.

- Uso de Enumeraciones: Para las características y el tipo de competidor, se eligió el uso de enum para tener un conjunto fijo y conocido de valores posibles, lo que facilita la validación y reduce errores en tiempo de ejecución.

- Separación de Menu y lógica de Juego: Se creó una clase Menu encargada de la interacción con el usuario, tanto entrada como salida, y validaciones básicas de datos de entrada. La clase 'Menu' no solo se limitó a facilitar la comunicación con el usuario, sino que también incluyó validaciones básicas de los datos de entrada, asegurando la integridad y coherencia de la información proporcionada por el usuario antes de pasarla a la lógica del juego. Esto contribuyó a una separación clara de responsabilidades y a un encapsulamiento apropiado de las responsabilidades de cada clase. No solo mejora la legibilidad y mantenibilidad del código, sino que también facilita futuras ampliaciones o modificaciones al sistema.

3. Persistencia de datos.

Lectura y Escritura de Archivos: Se decidió manejar la persistencia de datos a través de archivos planos, lo cual es una solución simple y suficiente para la escala del proyecto. Se descartó el uso de bases de datos u otros sistemas de almacenamiento más complejos.

4. Manejo de errores.

Excepciones Personalizadas: Se utiliza el manejo de excepciones para controlar errores de usuario y de flujo de programa, aunque se sugiere mejorar esta práctica utilizando excepciones personalizadas en lugar de genéricas para aumentar la claridad y facilitar la depuración.

5. Flexibilidad y mantenimiento.

Código Extensible y Mantenible: Se han implementado clases y métodos con la flexibilidad en mente, permitiendo la adición de nuevas características o cambios en las reglas del juego sin una reestructuración significativa del código. Se ha hecho uso del polimorfismo siempre que fue posible, para reutilizar

funcionalidad, tanto con asignaciones polimórficas, entidades polimórficas y estructuras de datos polimórficas.

Los textos de interacción con el usuario han sido internacionalizados usando text.properties, que, si bien por ahora sólo mantenidos en español, permiten muy fácilmente cambiar todos los textos que el usuario reciba a otro idioma.

Descripción de cada archivo.

- Caracteristica.java

Este archivo define un enum llamado Caracteristica, que representa las distintas características que un Competidor puede tener en el juego. Las enumeraciones definidas son VELOCIDAD, FUERZA, RESISTENCIA y DESTREZA. Estos valores se utilizan para evaluar las habilidades de los héroes y villanos en el juego.

- Competidor.java

Este archivo define una clase abstracta Competidor, que es una pieza central del modelo del juego. Esta clase implementa Comparable<Competidor> para establecer un orden entre los competidores y define una serie de métodos abstractos que deben ser implementados por las subclases.

Además, la clase Competidor proporciona métodos para la gestión del estado de un competidor respecto a su participación en una liga.

La clase Competidor es fundamental para la lógica del juego y sirve como base para clases concretas que representarán a los héroes, villanos y ligas dentro del juego.

- Juego.java

Este archivo define la clase Juego, que es el núcleo de la aplicación para el juego de héroes y villanos. Es responsable de coordinar las operaciones del juego y proporcionar la interfaz de usuario a través de la consola. La clase Juego maneja la lógica del menú principal, la administración de personajes y ligas, la realización de combates y la generación de reportes.

Características y Funciones Principales:

- Administración de Personajes y Ligas: Provee funcionalidades para cargar, crear, listar y guardar personajes y ligas desde y en archivos.
- Menú Principal y Submenús: Contiene métodos para mostrar el menú principal del juego y submenús para gestionar personajes y ligas, realizar combates y mostrar reportes.
- Realización de Combates: Facilita los combates entre personajes y ligas, permitiendo al usuario seleccionar competidores y las características por las cuales competirán.
- Reportes: Genera reportes basados en criterios seleccionados, como listar todos los competidores que pueden vencer a un competidor dado y listar competidores ordenados por características.

- Persistencia de Datos: Utiliza BufferedReader y BufferedWriter para leer y escribir datos en archivos, permitiendo así la persistencia de información entre sesiones de juego.
- Interfaz de Usuario: se encarga de llamar a una instancia de Menu para realizar la interacción con el usuario, tanto para entrada, validación básica de entrada, y salida.
- Manejo de Errores: Implementa manejo de excepciones para las operaciones que pueden fallar, como la lectura/escritura de archivos o la creación de competidores y ligas con datos no válidos.

La clase Juego actúa como el punto de entrada y el controlador del flujo del programa. Cada funcionalidad es accesible a través de una serie de menús en texto que guían al usuario a través de las opciones disponibles. Las instancias de Personaje y Liga se mantienen en listas y son manipuladas según las acciones del usuario. La clase se encarga de la lógica del juego y delega la presentación y la interacción con el usuario a una instancia de Menu.

- Liga.java

Este archivo define la clase Liga, que extiende la clase abstracta Competidor. Representa una liga que puede consistir en varios Personaje y otras Liga. La clase proporciona métodos para agregar competidores, calcular estadísticas promedio de sus competidores, y manejar la participación de la liga en combates.

- Main.java

Este archivo contiene la clase Main con el método main, que es el punto de entrada al programa. Instancia la clase Juego y comienza el ciclo principal que permite al usuario interactuar con el juego a través del menú principal.

- Menu.java

La clase Menu es responsable de interactuar con el usuario, mostrando opciones del menú, instrucciones y recolectando la entrada del usuario. Usa un ResourceBundle para internacionalizar el texto mostrado al usuario, permitiendo la localización del juego a diferentes idiomas y regiones.

- Ordenamiento.java

Este archivo define la clase Ordenamiento, que implementa la interfaz Comparator<Competidor>. Está diseñada para ordenar objetos Competidor (como Personaje y Liga) según un orden definido de características (Caracteristica). Tiene la capacidad de cambiar dinámicamente el criterio de comparación, lo que permite flexibilidad en cómo se pueden ordenar los competidores.

- Personaje.java

Este archivo define la clase Personaje, que extiende la clase abstracta Competidor. Representa a un personaje individual en el juego con un nombre real, un nombre de fantasía y un conjunto de características con valores numéricos específicos.

- TipoCompetidor.java

Este archivo define el enum TipoCompetidor, que se utiliza para diferenciar entre héroes y villanos dentro del juego. Es un tipo enumerado simple con dos posibles valores: VILLANO y HEROE.

- text.properties

Estos archivos contienen todos los textos que el usuario puede recibir en pantalla, internacionalizados según región/idioma. La clase Menu lee estos textos mediante un ResourceBundle. Actualmente sólo existe el archivo text_es_AR.properties pero es fácilmente mantenible a otro lenguaje.

Organización y distribución del trabajo.

Costa, Cristian Andres

- Initial Scaffolding del proyecto: creación de clases iniciales, métodos vacios y diagrama de clases. Definición de la estructura de directorios y configuración inicial del proyecto.
- Creación y mantenimiento del repositorio Git: configuración del repositorio en Git, establecimiento de estrategia de branching (Gitflow).
- Revisión de código: revisión de los pull requests, solución de conflictos de merge e integración del código.
- Testing y QA: reporte de bugs y seguimiento hasta su resolución.
- Informe.

Ocampo, Pablo

- Funcionalidad de combates, y métodos auxiliares asociados al menú combates.
- Creación de la clase Menu y su funcionalidad. Refactor para el encapsulamiento de toda la lógica de Juego referida a I/O en Menu con los cambios correspondientes.
- Pruebas integrales, validación de la lógica de todos los métodos de la clase Juego. Pequeños bugfixes para asegurar el correcto funcionamiento.
- Internacionalización de los textos.
- Revisión de código, comentarios explicativos y javadoc de las clases Juego y Menu. Reemplazo de métodos duplicados buscando reutilizarlos.
- Informe.

La Giglia, Rodrigo

- Funcionalidades de la clase juego para la creación de personajes y ligas y agregar competidores a una liga, creación de métodos adicionales para asegurar el correcto funcionamiento.
- Pruebas de funcionamiento de los métodos mencionados y relacionados, con lote de prueba y casos limite. Informe de bugs para correcciones de clases relacionadas y resolución de algunos de ellos.
- Actualización de UML.

López, Fernando

- Creación de clase Competidor, Personaje, y Liga con la encapsulación necesaria para el uso del usuario fuera de estas clases.
- Pruebas unitarias correspondientes a las clases anteriores.
- UML inicial.

Arias, Tomás

- Investigación sobre el uso de Comparable/Comparator
- Creación e implementación de la clase Ordenamiento
- Creación de los menús de los reportes
- Pruebas unitarias de la clase Ordenamiento
- Implementación de Comparable (compareTo) en la clase Competidor

Conclusión.

El desarrollo del juego "Héroes y Villanos" ha sido un emprendimiento integral que no solo resultó en la creación de un sistema de juego funcional y entretenido, sino que también fomentó un ambiente de colaboración y aprendizaje técnico. A través de desafíos de diseño e implementación, el equipo ha podido entregar un producto que alinea la lógica compleja del juego con una interfaz de usuario sencilla y accesible. Este proyecto ha demostrado la importancia de una arquitectura bien pensada y la eficacia de las prácticas de desarrollo ágil.

Incluso teniendo diagramada una planificación inicial detallada, surgieron cambios importantes a medio camino del desarrollo, decisiones de diseño que no habían sido contempladas. Estos cambios requirieron refactorizar funcionalidad, a lo que el equipo respondió dinámicamente, repartiendo responsabilidades y trabajando rápidamente para alcanzar los nuevos objetivos.

A pesar de los obstáculos encontrados, cada etapa del proyecto proporcionó oportunidades valiosas para mejorar nuestras habilidades en programación, gestión de proyectos y trabajo en equipo. La elección de priorizar un código limpio y mantenible sobre soluciones rápidas y temporales ha preparado el camino para futuras expansiones y mejoras del juego.

En resumen, "Héroes y Villanos" ha sido un proyecto satisfactorio que ha cumplido sus objetivos iniciales, y ha establecido una base sólida para el crecimiento continuo. Ha sido una confirmación de que un enfoque equilibrado entre una planificación cuidadosa y la adaptabilidad es esencial para el éxito en el desarrollo de software. Con la experiencia adquirida y las lecciones aprendidas, el equipo está bien equipado para enfrentar proyectos futuros con mayor confianza y eficiencia.