Práctica 5

Patrones de diseño

Inicio: Semana del 1 de abril
Duración: 3 semanas
Entrega: Semana del 22 de abril (fecha: cada grupo, el día siguiente a su última clase de prácticas y con hora límite las 23:55 h.)
Peso de la práctica: 30%

En el año 2941 de la Tercera Edad del Sol, a los lindes de Erebor, la Montaña Solitaria, Tierra Media, un ejército formado por criaturas de los Pueblos Libres – Hombres, Elfos y Enanos – se enfrentó a un ejército oscuro formado por Orcos y Huargos, en la que pasó a conocerse como la **Batalla de los Cinco Ejércitos**. La historia de este suceso se narra en "El hobbit" de J. R. R. Tolkien.

En esta última práctica de la asignatura se va a desarrollar en Java un simulador de la Batalla de los Cinco Ejércitos. Dicho simulador permitirá establecer:

- Las características de las **criaturas** enfrentadas (Hombres, Elfos, Enanos, Orcos, Huargos).
- El número, tamaño y tipo de criatura de las **tropas** de los **dos ejércitos** (Pueblos Libres y Oscuro). Nótese que este término "ejército" difiere del utilizado en el título de Batalla de Cinco Ejercitos. La batalla a simular siempre consta de dos "ejercitos", en los que puede participar hasta cinco tipos de criaturas.
- Las estrategias y reglas de ataque y defensa de los **ejércitos**, tanto para las tropas como para las criaturas.
- La condición de finalización de la **batalla**, considerando en un principio la aniquilación de todas las tropas de uno de los **dos ejércitos**.

Al margen de la temática y, por supuesto, con importancia prioritaria, el diseño e implementación del simulador se enfocará en los siguientes conceptos de Java y Programación Orientada a Objetos:

- Herencia y clases abstractas.
- Patrones de diseño.
- Interfaces Java.
- Interfaces Gráficas de Usuario (GUI) en Java.

Ejercicio 1. Clases para la creación y gestión de criaturas (4 puntos)

Sobre las características de las criaturas

Dentro del paquete `es.uam.eps.adsof.batalla5ejercitos.criaturas` se pide desarrollar las clases asociadas a los diferentes tipos y razas de criaturas, atendiendo a los requisitos que se explican a continuación.

Toda criatura tendrá los siguientes atributos:

- **Puntos de vida**. Un número entero que indica las heridas que la criatura puede soportar antes de morir. Al recibir una herida la criatura perderá un punto de vida.
- **Ataque**. Un número entero que indica la habilidad con la que la criatura ataca a otra.
- **Defensa**. Un número entero que indica la habilidad con la que la criatura se defiende de otra.
- **Heridas**. Un número entero que indica las heridas recibidas por la criatura en el último asalto (ataque + defensa).

Dependiendo de su **raza**, una criatura tendrá unos puntos de vida concretos y unos valores de fuerza y resistencia acotados. Estos se resumen en la tabla de abajo, donde [X, Y] significa que un atributo puede tomar valores enteros entre X e Y inclusive.

	Hombre	Elfo	Enano	Orco	Huargo
Puntos de vida	1	2	1	2	1
Ataque	[2,4]	[2,3]	[1,4]	[2,4]	[1,3]
Defensa	[1,3]	[2,3]	[1,2]	[1,2]	[2,4]

Además de métodos constructores, `get` y `toString`, las criaturas deberán tener los siguientes métodos:

- **`boolean estaMuerto()`**, que devuelve `true` si la criatura tiene 0 puntos de vida, y `false` en caso contrario.
- **`void atacar(Criatura oponente)`**, que atendiendo a la habilidad de ataque de la criatura, a la habilidad de defensa de un oponente y a una tirada de dado (valor aleatorio), causa o no causa una herida al oponente. Al final del enunciado se dan más detalles de cómo se realiza un asalto (ataque-defensa).
- **`void addHeridas(int numeroHeridas)`**, que incrementa `numeroHeridas` heridas a la criatura.
- **`void aplicarHeridas()`**, que aplica las heridas recibidas por la criatura en el último asalto, es decir, resta a los puntos de vida las heridas recibidas en el último asalto; y posteriormente deja a 0 las heridas para el asalto siguiente. Nota: la gestión de heridas y la declaración de este método pueden ser otras.

La jerarquía de clases asociada a las criaturas debe permitir la distinción entre **tipos de razas**: "criaturas libres" (instancias de las clases Hombre, Elfo y Enano) y "criaturas oscuras" (instancias de las clases Orco y Huargo). Como se verá más adelante, un ejército estará formado por criaturas pertenecientes a razas de un mismo tipo: libres u oscuras.

Atendiendo a "El Silmarillion" de J. R. R. Tolkien, elfos y orcos son considerados los "Primeros Nacidos", al ser los primeros hijos de Eru, el único, el creador y dios absoluto de Arda (el mundo donde se encuentra Tierra Media). En la simulación a realizar se considera que estas dos razas tienen una característica especial por la cual son capaces de curar ciertas heridas de forma inmediata. Esta característica se implementará a través de un método adicional llamado `curarHerida` que ha de ser exigido a aquellas razas (podría haber otras distintas a elfos y orcos) que tuviesen el citado don "divino" de curación. La implementación específica del método `curarHerida` para elfos y orcos consistirá en que con una probabilidad dada se recupera una herida. Dicha probabilidad será 0.3 y 0.2 para elfos y orcos respectivamente.

Sobre la creación de criaturas

Para la creación de criaturas en el simulador se pide diseñar y desarrollar clases siguiendo el patrón de diseño **Factory method**.

El patrón de diseño **Factory method** proporciona una interfaz para crear objetos de subclases de una clase dada o de clases que implementan una *interface* dada. En concreto, para la creación de criaturas, se pide usar una *interface* `CriaturaFactoria` que obligue a implementar un método `crearCriatura`:

```
public interface CriaturaFactoria {
	public Criatura crearCriatura();
}
```

Para cada raza se debe desarrollar una clase que implemente `CriaturaFactoria` y que cree instancias de la clase asociada a la raza, como por ejemplo `ElfoFactoria` y `OrcoFactoria`. Estas factorías tendrán en cuenta las características de puntos de vida, y valores acotados de ataque y defensa dados en la tabla de arriba.

Ejercicio 2. Clases para la creación y gestión de ejércitos (4 puntos)

Dentro del paquete `es.uam.eps.adsof.batalla5ejercitos.ejercitos` se pide desarrollar las clases asociadas a ejércitos y tropas, atendiendo a los requisitos que se explican a continuación.

En el simulador se debe considerar que un ejército está formado por varias tropas de un mismo tipo, y que cada tropa está formada por criaturas de una misma raza. Así, por ejemplo, el ejército de los Pueblos Libres podría estar formado por una tropa de 300 hombres, una tropa de 100 elfos, y dos tropas de 100 y 150 enanos respectivamente.

Sobre la creación de tropas

Según lo anterior una tropa debe tener una lista guerreros de una raza dada. Para llevar esto a cabo se pide desarrollar una clase `Tropa` cuyo constructor reciba una factoría que cree las criaturas de la tropa. El constructor también recibirá como argumento de entrada el número de guerreros de la tropa. Su declaración quedaría como sigue:

```
public Tropa(CriaturaFactoria factoria, int numGuerreros)
```

Aparte del constructor, se pide que la clase `Tropa` al menos tenga los siguientes métodos:

- **`public boolean estaAniquilada()`**, que devuelve `true` si todos los guerreros de la tropa están muertos, y `false` en caso contrario.
- **`public void atacar(Tropa oponente)`**, que hace que cada guerrero de la tropa `this` ataque a otro guerrero de la tropa oponente elegido de manera aleatoria. Nótese que esto puede dar lugar a que varios guerreros ataquen a uno dado en la tropa oponente (tal vez causándole más de una herida en el asalto).
- **`public void aplicarHeridas()`**, que aplica las heridas recibidas a todas las criaturas de la tropa en el último asalto. Nota: la declaración de este método puede ser otra.

Sobre la creación de ejércitos

Se pide que la creación de un ejército se haga de una sola vez pasando a un método de una clase `Ejercito` la descripción de las tropas (raza y número de sus guerreros) a través de un `Map`, que almacene los números de guerreros de las tropas de las distintas razas que formarán ese ejército. Por ejemplo, la siguiente descripción:

Hombre	{200, 100}
Elfo	{50}
Enano	{150}

representa un ejército con dos tropas de 200 y 100 hombres, una de 50 elfos y otra de 150 enanos.

En el `Map` anterior habrá que decidir qué clases de objetos se guardan como claves. Para ello habrá que tener en cuenta la manera en la que se crean criaturas y tropas.

Adicionalmente, como se comentó en el primer ejercicio, un ejército sólo puede tener tropas de razas de un único tipo: de criaturas libres (Hombres, Elfos o Enanos) o de criaturas oscuras (Orcos o Huargos). Para satisfacer este requisito se pide desarrollar la jerarquía de clases ejército adecuada e implementar un control de excepciones al respecto.

Ejercicio 3. Clase para la simulación de la batalla (1.5 puntos)

Dentro del paquete `es.uam.eps.adsof.batalla5ejercitos` se pide desarrollar la clase `Batalla` que implemente la simulación de la Batalla de los Cinco Ejércitos. Dicha simulación se deberá realizar en un método simular que podría seguir el pseudocódigo de abajo. Se deja libertad para incorporar otros métodos diferentes con funcionalidades específicas.

```
Crear Ejército de los Pueblos Libres
Crear Ejército Oscuro
Mientras (fin de batalla = false)
       Visualizar estado de la batalla (ejércitos)
       // Realizar asalto
       Ejército de los Pueblos Libres ataca al Ejército Oscuro
       Ejército Oscuro ataca al Ejército de los Pueblos Libres
       // Aplicar daños
       Aplicar daños al Ejército de los Pueblos Libres
       Aplicar daños al Ejército Oscuro
Visualizar ejército vencedor
```

El **ataque** de un ejército a otro se puede hacer de muchas maneras. Se permite que sea tan simple como que cada tropa del ejército atacante ataque a una tropa oponente elegida de forma aleatoria. Se recuerda que el ataque entre tropas es similar y está descrito en el ejercicio 2. El ataque entre criaturas puede seguir el siguiente pseudocódigo:

```
dado1 ← Número aleatorio entre 1 y 6 (inclusive)
dado2 ← Número aleatorio entre 1 y 6 (inclusive)
Si (dado1 + ataque del atacante > dado2 + defensa del oponente)
       Añadir una herida al oponente
```

Nótese que para elfos y orcos a este pseudocódigo debe incorporarse la característica de curar heridas descrita en el ejercicio 1.

En Java la generación de números aleatorios se realiza a través de una instancia de la clase `java.util.Random`, que puede crearse como sigue:

```
Random r = new Random(Calendar.getInstance().getTimeInMillis());
```

donde el tiempo de sistema actual en milisegundos se utiliza como semilla del generador de números aleatorios.

Ejercicio 4. Modificación de las características de las criaturas (0.5 puntos)

Una vez concluida la implementación y comprobada la correcta ejecución del simulador, se pide extender este último con nuevas características para elfos y orcos sin necesidad de perder en el código las características iniciales. En concreto, se pide considerar la existencia de **Elfos Noldor** y **Orcos Uruk-hai** como criaturas de las razas Elfo y Orco, respectivamente, que poseen las siguientes características:

	Elfo Noldor	**Orco Uruk-hai**
Puntos de vida	2	2
Ataque	[3,4]	[3,5]
Defensa	[3,4]	[2,3]

Además del código necesario para esta modificación, en la memoria de la práctica se debe incluir un apartado donde se dé respuesta a las siguientes preguntas:

- ¿Qué se ha cambiado del código para considerar las nuevas características de elfos y orcos?
- ¿En qué medida el patrón de diseño *Factory method* ha permitido añadir tales características de forma flexible?

Ejercicio optativo. Interfaz Gráfica de Usuario (1 punto extra)

Como parte optativa de esta práctica se establece la implementación de una interfaz gráfica de usuario en la que se visualice paso a paso el desarrollo de la simulación de la batalla.

Normas de entrega

Se debe entregar:

- un directorio src con todo el código Java,
- un directorio doc con la documentación generada,
- un archivo PDF con una memoria explicativa de la realización de la práctica de no más de 10 páginas que contenga información adicional a la documentación generada con javadoc, describiendo el diseño, estructura y particularidades de la aplicación desarrollada, justificando las decisiones que se hayan tomado la estructura de la código, los problemas principales que se han abordado y cómo se han resuelto, así como los problemas pendientes de resolver. La memoria debe contener el diagrama de clases y una explicación de las pruebas realizadas.

Lo anterior se debe añadir a un único fichero ZIP que deberá llamarse de la siguiente manera: GR<numero_grupo>_<nombre_estudiantes>.zip. Por ejemplo Marisa y Pedro, del grupo 2261, entregarían el fichero: GR2261_MarisaPedro.zip.