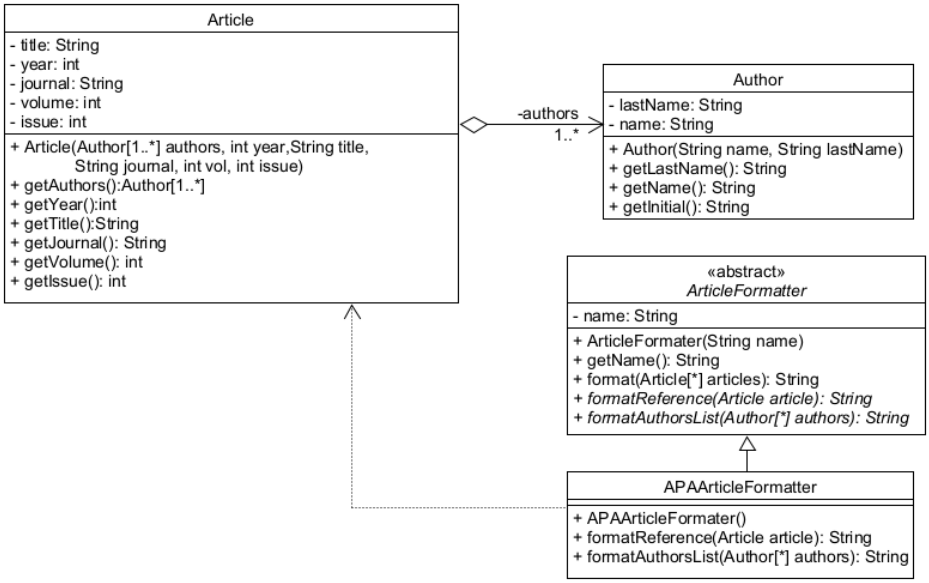
Análisis y Diseño de Software
Curso 2023-2024
Práctica 2
Diseño Orientado a Objetos

Inicio: A partir del 12 de febrero.
Duración: 2 semanas.
Entrega: En Moodle, una hora antes del comienzo de la siguiente práctica según grupos (semana del 26 de febrero)
Peso de la práctica: 15%

El objetivo de esta práctica es introducir los conceptos básicos de orientación a objetos, desde el punto de vista del diseño (usando UML), así como de su correspondencia en Java.

Apartado 1 (2.5 puntos):

Se pide diseñar una aplicación que, a partir de una base de datos de artículos científicos, permita imprimir la información de los artículos en diferentes formatos (por ejemplo, IEEE o APA, entre otros). Para ello se parte del diagrama de clases UML proporcionado.



Cada artículo (clase `Article`) de la aplicación contiene atributos como el título, el año de publicación, la lista de autores, la revista en la que está publicado (`journal`), el volumen (`volume`) y su número (`issue`). Para almacenar la información de cada autor, se dispone de la clase `Author`, que almacena el nombre y apellido (`name` y `lastName`) del autor, tiene métodos `getter` para acceder a dichos valores, y un método `getInitial`() que devuelve la inicial del nombre del autor.

La aplicación requiere poder imprimir la información de los artículos en distintos formatos, con facilidad para la extensión con nuevos formatos. Para ello, se hace uso de clases formateadoras que utilizan los datos del artículo para generar un `String` determinado. Las clases formateadoras heredan de la clase abstracta `ArticleFormatter`, que tiene como atributo su nombre (por ejemplo, APA o IEEE), y tiene su correspondiente método getter (`getName():String`). El constructor recibe por parámetro su nombre para inicializar el atributo. Además, cuenta con dos métodos abstractos: 1) `formatReference(Article: article): String`, que recibe por parámetro un artículo y produce un String con la salida formateada, y 2) `formatAuthorsList(Author[*] authors): String`, que recibe una colección de autores y lo formatea del modo adecuado. Finalmente, el método `format` recibe una colección de artículos y devuelve un `String` con todos los artículos formateados.

Cada formateador concreto heredará de esta clase `ArticleFormatter` y proporcionará una implementación específica para cada uno de los métodos abstractos con el fin de dar un formato concreto de acuerdo con el estilo correspondiente. En el diagrama se dispone de una clase formateadora para el formato APA (clase `APAArticleFormatter`). Así, el formato APA presentará las referencias de artículos siguiendo el siguiente formato:

Apellido, Inicial. (año). Título del artículo. Nombre de la revista, Volumen (Issue).

A modo de ejemplo:

Kaczorek, T. (2016). Minimum energy control of fractional positive electrical circuits. Archives of Electrical Engineering, 65(2).

Como ya sabes, los constructores tienen el mismo nombre que la clase, y se invocan mediante el operador `new` para construir un objeto (crear una instancia de la clase) e inicializar sus atributos. Por ejemplo, el constructor de `ArticleFormatter` recibe un parámetro e inicializa el atributo `name`. El constructor de la clase `APAArticleFormatter` llama al constructor de la superclase `ArticleFormatter` mediante `super(...)`, y luego podría ejecutar código adicional. La anotación opcional *`@Override`* indica que la intención del programador es sobrescribir un método de alguna superclase, y así el compilador puede comprobar que el método tiene igual firma que el método sobrescrito.

Nota: Los diagramas de clases son abstracciones del código, así que normalmente y para facilitar su comprensión, no se suelen incluir todos los detalles necesarios para la codificación. En concreto, se suelen omitir los métodos `setters`, `getters` y los constructores.

El siguiente listado proporciona una implementación para el diagrama de clases (cada clase va en un fichero .java que se llama como la clase), así como una clase `Main` con un `main` de ejemplo y su salida (disponibles en Moodle):

```
import java.util.List;

public class Article{
	private List<Author> authors;
	private int year, volume, issue;
	private String title, journal;

	public Article(List<Author> authors, int year, String title, String journal,
			int vol, int issue) {
		this.authors = authors;
		this.year = year;
		this.title = title;
		this.journal = journal;
		this.volume = vol;
		this.issue = issue;
	}
	public List<Author> getAuthors() { return authors;  }
	public int    getYear()    { return year;    }
	public String getTitle()   { return title;   }
	public String getJournal(){ return journal;}
	public int    getVolume()  { return volume;  }
	public int    getIssue()   { return issue;   }
}
```

```
public class Author {
	private String lastName;
	private String name;

	public Author(String name, String lastName) {
		this.name = name;
		this.lastName = lastName;
	}
	public String getLastName() { return lastName; }
	public String getName()     { return name; }
	public String getInitial()  { return name.substring(0, 1).toUpperCase(); }
}
```

```
import java.util.List;

public abstract class ArticleFormatter {
	protected String name;

	public ArticleFormatter(String formatterName) { this.name = formatterName;}
	public abstract String formatAuthorsList(List<Author> authors);
	public abstract String formatReference(Article a);
	public String getName() { return this.name; }
	public String format(List<Article> articles) {
		String result = "";
		for (Article a : articles)
			result+= this.formatReference(a)+"\n";
		return result;
	}
}
```

```
import java.util.List;

public class APAArticleFormatter extends ArticleFormatter {
	public APAArticleFormatter() { super("APA"); }
```

```
	@Override public String formatAuthorsList(List<Author> authors) {
		StringBuffer sb = new StringBuffer();
		for (Author a : authors) {
			sb.append((sb.length()>0)? ", " : "");
			sb.append(a.getLastName() + ", "  + a.getInitial() + ".");
		}
		return sb.toString();
	}
	@Override public String formatReference(Article a) {
		return formatAuthorsList(a.getAuthors()) + " "+
			"(" + a.getYear() + "). " +
			a.getTitle() + ". " + a.getJournal() + ", " +
			a.getVolume() + "(" + a.getIssue() + ").";
	}
}
```

```
import java.util.List;

public class Main {
  public static void main(String[] args) {
	Article Kaczorek2016 = new Article(
			List.of(new Author("Tadeusz", "Kaczorek")),                          // author
			2016,                                                                // year
			"Minimum energy control of fractional positive electrical circuits", // title
			"Archives of Electrical Engineering",                                // journal
			65, 2); // vol and issue

	Article Uchiyama2014 = new Article(
			List.of(new Author("Satoru", "Uchiyama"),
				new Author("Atsuto", "Kubo"),
				new Author("Hironori", "Washizaki"),
				new Author("Yoshiaki", "Fukazawa")),                     // authors
			2014,                                                        // year
			"Detecting Design Patterns in Object-Oriented Program Source Code "+
			"by Using Metrics and Machine Learning",                     // title
			"Journal of Software Engineering and Applications",          // journal
			7, 12); // vol and issue

	List<Article> references = List.of(Kaczorek2016, Uchiyama2014);
	List<ArticleFormatter> formatters = List.of( new APAArticleFormatter(),
	                                             new IEEEArticleFormatter()); // add this class

	for (ArticleFormatter formatter : formatters) {
		System.out.println("Articles in "+formatter.getName()+" format:");
		System.out.println(formatter.format(references));
	}
  }
}
```

Salida esperada:

```
Articles in APA format:
Kaczorek, T. (2016). Minimum energy control of fractional positive electrical circuits. Archives of Electrical Engineering, 65(2).
Uchiyama, S., Kubo, A., Washizaki, H., Fukazawa, Y. (2014). Detecting Design Patterns in Object-Oriented Program Source Code by Using Metrics and Machine Learning. Journal of Software Engineering and Applications, 7(12).

Articles in IEEE format:
T. Kaczorek, "Minimum energy control of fractional positive electrical circuits", Archives of Electrical Engineering, vol. 65, no. 2, 2016.
S. Uchiyama, A. Kubo, H. Washizaki, Y. Fukazawa, "Detecting Design Patterns in Object-Oriented Program Source Code by Using Metrics and Machine Learning", Journal of Software Engineering and Applications, vol. 7, no. 12, 2014.
```

Se pide: Extender el **diseño UML** y el **programa Java** para conseguir la salida deseada, en la que, como puede apreciarse, no solo se formatean los artículos siguiendo el estilo APA, sino también el formato IEEE. Para ello, deberás añadir una nueva clase formateadora llamada `IEEEArticleFormatter`, que dará forma a la salida con la siguiente estructura:

Inicial. Apellido, "Título del artículo", Nombre de la revista, vol. Volumen, no. Issue, año.

Por ejemplo: T. Kaczorek, "Minimum energy control of fractional positive electrical circuits", Archives of Electrical Engineering, vol. 65, no. 2, 2016.

Nota: Las clases del enunciado no declaran un paquete (usan el paquete por defecto), lo que es una mala práctica. Más adelante aprenderás a usar paquetes, y su utilidad.

Apartado 2 (4 puntos)

Se busca diseñar una biblioteca digital de libros electrónicos online que facilite a escritores la publicación de sus obras para que los lectores puedan acceder a ellos en línea. La plataforma contará con dos tipos de usuarios: escritores y lectores. Cada usuario tendrá un identificador interno único, su nombre completo, un nombre de usuario, una contraseña para autenticación, y una o más tarjetas de crédito (con número, un código de verificación de 3 cifras, y fecha de caducidad) en las que realizar los ingresos y cargos según corresponda. Para poder operar en la plataforma, esta deberá permitir dar de alta a los diferentes tipos de usuario, y que estos puedan validar sus credenciales con nombre de usuario y clave.

Los escritores proporcionarán un número de identificación fiscal, un porcentaje de comisión por cada descarga de sus libros por parte de los lectores, y un nombre comercial o pseudónimo literario. Para registrarse, un escritor debe tener al menos un libro que publicar. La plataforma podrá acceder a los libros de cualquier escritor registrado en cualquier momento.

Por su parte, los lectores tendrán un plan de precios asociado. Los planes pueden ser: Básico (sin tarifa mensual, se paga sólo por los libros adquiridos), Estándar (tarifa mensual de 9.99€, con un descuento del 50% en todas las compras), o Premium (tarifa mensual de 19.99€, con acceso a todo el contenido sin coste adicional).

La plataforma deberá tener acceso al registro de los libros comprados por los lectores, y calcular el importe mensual a pagar por cada uno de estos, atendiendo al plan de precios al que se encuentra abonado. El cálculo del importe mensual debe contemplar todos los libros comprados por el lector ese mes, más la cuota mensual de subscripción según su plan, más los actos promocionales a los que se ha apuntado (como se detalla más abajo).

Los libros tendrán un identificador único, un título, un precio, uno o más géneros (p.ej. novela narrativa, ensayo, novela gráfica, etc.), el año de publicación, los autores involucrados, los premios obtenidos (si los hay), la extensión en páginas, y los puntos que se obtienen por su compra (ver más abajo). Además, tendrán una serie de puntuaciones en una escala del 0 al 5, que los lectores podrán haber otorgado a los libros que han comprado. Debe ser posible acceder eficientemente a todos los libros de un cierto género.

Se incluirá también la opción de comprar libros de forma individual o por sagas completas, donde cada saga contenga varios libros que comparten personajes o el argumento principal. Ejemplos de sagas conocidas incluyen "Harry Potter", "La Trilogía del Baztán" o "El Señor de los Anillos". Las sagas también tendrán un identificador único, un título, un precio (que vendrá dado como la suma del precio de los libros que la componen al que se le puede aplicar un descuento si se compra la saga completa), los géneros (que viene dado por la unión de los géneros de los libros que componen la saga), el año (que debería corresponder con el año de publicación del primer libro que la compone), los autores involucrados, los premios, la extensión en páginas (que será la suma de las páginas de todos los libros que la componen), y los puntos acumulados por su compra (que es la suma de los puntos de los libros que la componen). Al igual que los libros, los lectores también pueden dar una puntuación del 0 al 5 a las sagas que han comprado.

Para favorecer las campañas de márquetin, los lectores podrán acumular puntos por la compra de libros. Estos se podrán canjear más adelante en actos promocionales exclusivos con sus escritores favoritos y que también podrán comprar a través de la plataforma. En concreto, los actos promocionales pueden ser: lecturas en abierto, firmas de libros, encuentros virtuales para discutir la obra, y talleres de escritura creativa. En la plataforma, estos actos tendrán una fecha de celebración, un precio (que se cobra a los lectores en la mensualidad correspondiente a la fecha de celebración), el descuento que pueden tener quienes tengan un número concreto de puntos (también a especificar en el acto), una descripción textual del lugar de celebración, y el autor con el que están vinculados.

Se pide:

a) Realiza un diagrama de clases UML que refleje el diseño de la aplicación. No incluyas constructores, getters o setters, a no ser que se pidan en el próximo apartado **(2.5 puntos)**.

b) Añade en las clases los métodos y atributos necesarios para realizar las siguientes funciones **(1 punto)**:

1. Realizar *login* de un usuario en la aplicación.
2. Calcular el adeudo mensual de un lector.
3. Obtener todos los libros y sagas comprados por un lector.
4. Obtener el precio, año, géneros, autores, premios, páginas y puntos de un libro o una saga.

Ten en cuenta que en algún caso puede ser necesario añadir métodos auxiliares. No es necesario que incluyas el código del método.

c) Proporciona pseudocódigo (o código Java) para el/los métodos que calculan el adeudo mensual de un lector **(0.5 puntos)**.

Apartado 3 (3.5 puntos):

Se quiere diseñar un videojuego para uno o varios jugadores, cada uno de los cuales controlará un personaje en un mundo virtual en 2D. Los personajes deben enfrentarse con enemigos, que tienen un comportamiento que controla la aplicación. Al empezar el juego, cada personaje se configura con un nombre y una imagen de avatar, y el juego les asigna aleatoriamente una velocidad de desplazamiento y una fuerza de ataque. Los enemigos también tienen una velocidad y fuerza de ataque. Durante el juego, tanto enemigos como personajes tendrán una posición en el mundo virtual, y unos puntos de vida.

Consideraremos dos tipos de enemigos: estándares y jefes, de los cuales podrá haber múltiples instancias durante el juego. Ambos tipos de enemigo tienen un grito de guerra configurable, así como su radio de acción (la distancia a la que pueden atacar a un personaje). El jefe es en todo igual al estándar, pero tiene una ratio de recuperación de vida, que le permite ir recuperándose en cada ciclo del juego.

El comportamiento cualquier tipo de enemigo viene dado por una máquina de estados, que está formada por estados y transiciones. Cada estado incluye información sobre el nivel de daño de sus ataques (es decir, un multiplicador del daño base), la decisión de si debe atacar o no a cualquier personaje que se encuentre en su posición, si debe moverse (al norte, sur, este u oeste; o bien alejarse o acercarse al personaje más cercano dentro de su radio de acción) o quedarse quieto. Desde un estado se podrá pasar a otro si se ha definido una transición del primer estado al segundo estado, y se cumple una condición que depende del tipo de transición. En particular, consideraremos cuatro tipos de transiciones, que se disparan cuando: el nivel de vida sea mayor o igual que cierto valor; menor que cierto valor; exista cierto número de personajes o más en su radio de acción; o menos de cierto número.

El juego consiste en la ejecución de una secuencia de ciclos. En cada ciclo, tanto los personajes como los enemigos pueden moverse hacia uno de los puntos cardinales o no moverse, así como atacar o no. En el caso de los personajes, el movimiento y la decisión de atacar o no se obtiene de un objeto de tipo "Control" (que asumimos lee la entrada del teclado), que cada personaje tiene asociado. Dicha clase tendrá un método "obtenerMovimiento" que devuelve la dirección en que se mueve el personaje, y un método "atacar" que devuelve si el personaje quiere atacar. El movimiento de los enemigos y su decisión de atacar vendrá dado por el estado actual en el que se encuentran. En cada ciclo de juego, cada enemigo puede cambiar su estado actual, de acuerdo al comportamiento definido en su máquina de estados. Para tomar la decisión de dónde moverse, el juego proporciona la colección de personajes y enemigos que se encuentran en el radio de acción. Además, en cada ciclo, los personajes perderán 0,1 puntos de vida, los enemigos estándar no pierden vida, y los jefes podrán recuperar vida según la ratio de recuperación que definan. Los enemigos estándar y los personajes no recuperan puntos de vida. Cuando un personaje o enemigo llega a 0 puntos de vida, desaparece del juego. El juego termina cuando todos los personajes desaparecen.

Se pide:

a) El diseño del videojuego usando diagramas de clases UML. Añade los métodos necesarios en las clases para obtener la funcionalidad que menciona el enunciado. (**2.5 puntos**)

b) Crea un diagrama de objetos (**1 punto**) que modele un enemigo estándar que inicialmente tiene 100 puntos de vida, velocidad 2, y fuerza 5, así como un comportamiento en el que:

- Inicialmente está en estado "EnReposo", donde no ataca, su multiplicador de daño es 0, y no se mueve.
- Cuando detecta un personaje en su radio de acción, pasa al estado "Ataque", que tiene un multiplicador de daño 2, y se acerca al personaje.
- En el estado "Ataque", si detecta que su nivel de vida es menor de 50, el estado pasa a "Huida", donde se aleja del personaje y no ataca.
- En el estado "Huida", cuando detecta que no hay personajes, vuelve al estado "EnReposo".

Normas de Entrega:

- Se deben entregar los apartados 1, 2 y 3. Crea un directorio por cada apartado.
- La entrega la realizará uno de los estudiantes de la pareja, a través de Moodle.
- Si el ejercicio pide código Java, además del código, se debe entregar la documentación generada con *javadoc*. Si el ejercicio pide un diagrama de diseño, se debe entregar en PDF junto con una breve explicación (dos o tres párrafos a lo sumo) del mismo.
- Se debe entregar un único fichero ZIP / RAR con todo lo solicitado, que deberá llamarse de la siguiente manera: `GR<numero_grupo>_<nombre_estudiantes>.zip`. Por ejemplo, Roberto y Carlos, del grupo 2261, entregarían el fichero: `GR2261_RobertoCarlos.zip`.