Instituto Tecnológico de Costa Rica

Ingeniería en Computadores

Algoritmos y estructuras de Datos II

II Extraclase

Patrones de diseño

Profesor:  
Luis Diego Noguera Mena

Alumnos:  
Brian Wagemans Alvarado

Alejandro Vásquez Oviedo

Tabla de Contenidos

1. Resumen de artículos encontrados ………………… 1
   1. Resumen de los artículos …. 1
   2. Design patterns impact on software quality: Where are the theories? ……………………. 2
2. Patrones de diseño ………….. 3
   1. Abstract Factory ……………. 3
      1. Proposito ………………... 3
      2. Diagrama de Clases …….. 3
      3. Estructura……………….. 3
      4. Contexto………………… 3
   2. Adapter……………………… 4
      1. Proposito ………………... 4
      2. Diagrama de Clases …….. 4
      3. Estructura……………….. 4
      4. Contexto………………… 4
3. Bibliografía …………………. 5

1. Resúmenes de artículos encontrados

A. One Approach to Improve the Software Quality by Applying Software Design Patterns

La industria del software reconoce la importancia de la alta calidad, confiabilidad y soporte. Siempre se requieren soluciones efectivas que mejoren dicha calidad. Los patrones de diseño son una excelente opción para lograr esto, además de permitir la reutilización de código y la mantenibilidad.

Los patrones de diseño además de ser por sí mismos un método de solución a un problema general pueden combinarse entre sí para dar respuestas a problemáticas más complejas. La decisión al combinar patrones debe realizarse tomando en cuenta las relaciones, similitudes, superposición, interacción y asistencia.

El artículo plantea el desarrollo de un sistema utilizando combinaciones de patrones de diseño (ABCDP) para obtener un producto final seguro y que cumpla con los estándares de calidad de la industria. Donde se requieren distintas tareas y con los criterios mencionados anteriormente se realizan las conexiones entre patrones que brinden una solución.

Dentro de los patrones utilizados se encuentran el Command que se utilizó en la implementación de un menú y un submenú. Un facade en la simplificación de varios procesos como el envío de e-mails (que incluye recuperación de contraseña), generar reportes de distintos formatos, realizar distintos cálculos. Un singleton para generar un enlace global a la base de datos. Factory Method para la encriptación de datos y su modificación.

El resultado indicado en el artículo tras la implementación combinada de los mencionados patrones es un incremento en la mantenibilidad, indica, además, que los criterios de calidad no están separados entre sí, por tanto, al mejorar la mantenibilidad mejoró la confiabilidad, la capacidad de prueba y la flexibilidad, siendo una métrica directa de las dos últimas mencionadas la reutilización también incrementó considerablemente. Con la mejora de estos elementos también mejoró la portabilidad, y así continuó con las demás métricas. Siendo que la investigación demuestra que el uso combinado de distintos patrones de diseño asociados de manera lógica resulta en la mejora lineal de las diferentes métricas utilizadas en la medición de calidad del software. [1]

B. Design patterns impact on software quality: Where are the theories?

La programación orientada a objetos tiene clases recurrentes a lo largo de varios problemas, a esto se le llaman patrones de diseño. Los programadores expertos los reutilizan, ya que han sido probados y además vuelven al programa flexible. Todos los usos de estos patrones tienen cosas en común, estas son: el nombre, que describe el problema a resolver, este a su vez es, cuando aplicar el patrón. La solución, que es la implementación del patrón en el software, su relación, responsabilidad y colaboración con el resto del sistema.

Los patrones se pueden clasificar en de 2 maneras: por su propósito (creación, estructural, comportamiento) o su ámbito (clase u objeto). Algunos de estos patrones son Facade, Adapter, Singleton.

El Facade simplifica el acceso a muchas clases haciendo que todo acceso a estas clases provenga de esta. Por lo que cualquier tipo de interacción entre las clases y el cliente ocupa pasar por él. Esto funciona ya que el cliente no ocupa saber que clases existen y estas clases se pueden cambiar fácilmente desde el facade sin afectar a los clientes. Por esta razón se clasifica bajo objetos y estructural.

El adapter sirve para cambiar la interfaz de un objeto y así volverlo compatible con el cliente esperado, este se puede implementar en la clase o en tiempo de ejecución. Por lo tanto, es clasificado bajo estructural, objetos y clase. Esto puede ser altamente útil ya que se pueden transformar objetos similares que no funcionan inicialmente juntas en un conjunto.

Singleton tiene el propósito evitar que se creen varias instancias para lograr un acceso único a la instancia. Esto es necesario cuando un recurso dentro de la clase se ocupa centralizar, además de que una variable global no nos garantiza que un objeto solo se cree una vez en el programa. La clave para lograr este comportamiento es crear un método estático que devuelve la única instancia del objeto y haciendo que el constructor sea privado. También se debe mencionar que esta clase debe declararse final, de caso contrario se podrían crear clases hijas y crear más instancias del objeto.

Factory Method es un patrón cuya utilidad se encuentra en separar la creación del objeto de su jerarquía, tiene varias ventajas entre estas: Centraliza la creación del objeto, hace que el sistema se pueda centralizar. Esto trae el beneficio que se logra abstraer el usuario de la instanciación del objeto.

Resumiendo, los patrones de diseño tienen diferentes tipos de usos y vuelven el código en algo manejable a futuro, por esta razón se deberían utilizar siempre que se dé la oportunidad. [2]

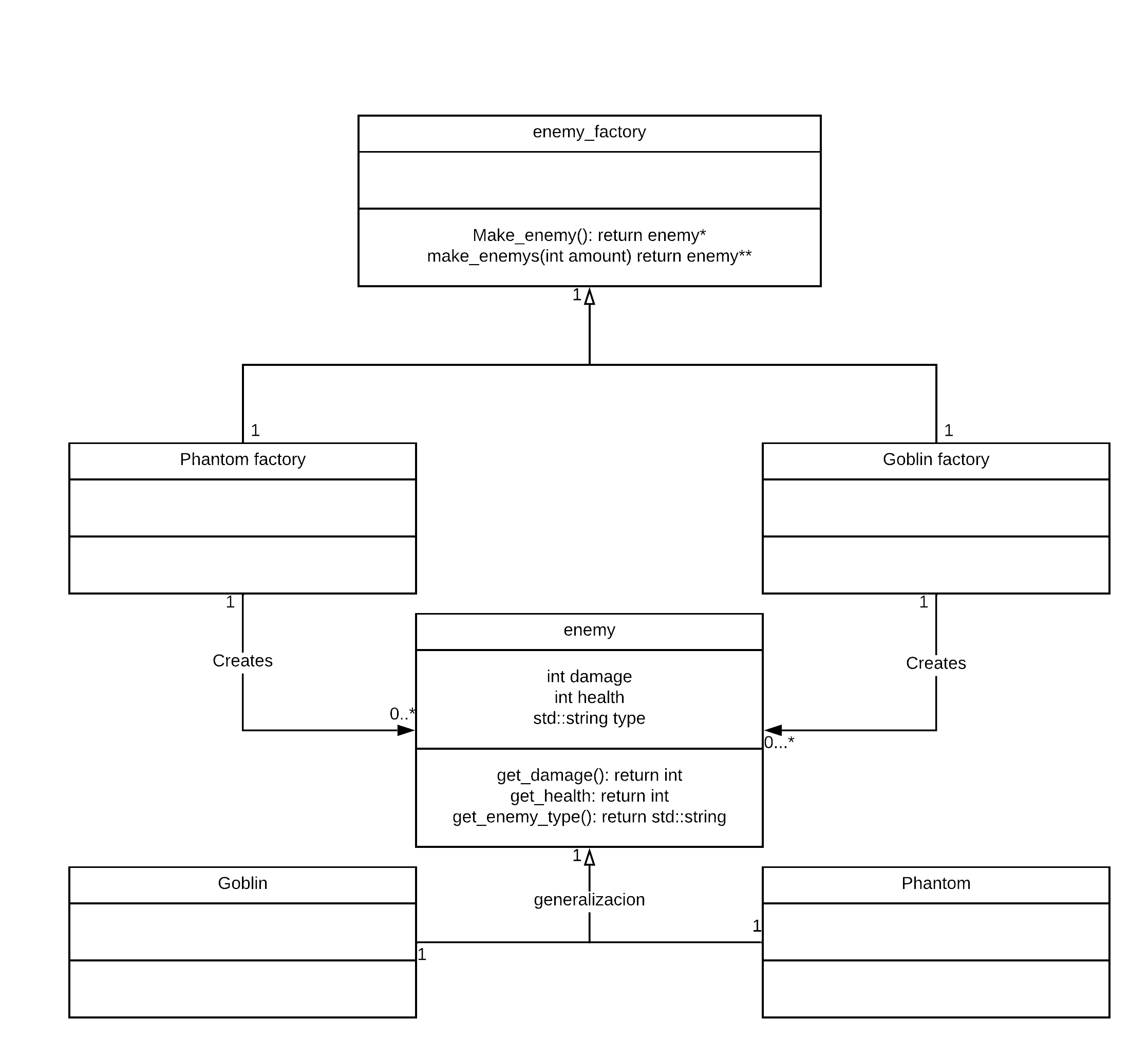
2. Patrones de diseño

A. Abstract Factory

I. Propósito

Crea una manera de crear una familia de objetos, sin especificar una clase concreta.

II. Diagrama de clases

Fig. 1. Diagrama de clases del patrón Abstract Factory utilizado en el ejemplo programado

III. Estructura

cómo se puede notar de este diagrama UML se tiene una clase padre llamada enemy\_factory cuyo trabajo es abstraer el trabajo de las distintas fábricas de enemigos, estas fábricas tienen como función crear al enemigo respectivo, por ejemplo, goblin\_factory crea goblins. El mayor se puede apreciar al ver que se abstraen los enemigos a su clase padre enemy, de esta manera se puede aprovechar el polimorfismo.

IV. Contexto

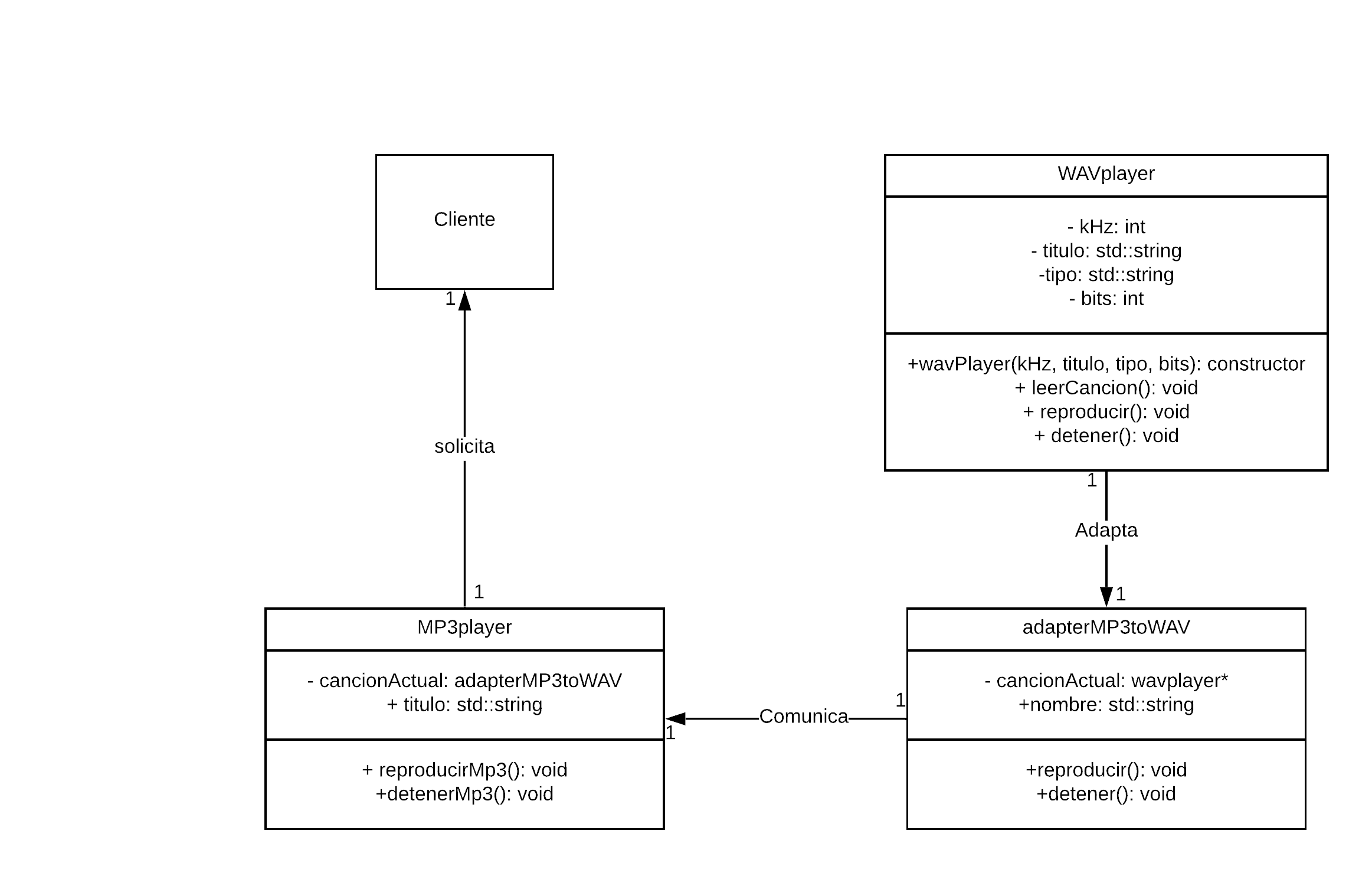
1. Para crear enemigos en un videojuego, no se ocupa saber qué clase de enemigo es para poder interactuar con él ya que tienen funciones básicas, como una vida concreta, un nivel de ataque, etc.
2. La creación de objetos genéricos cuyo comportamiento específico no importa en ese punto del programa. Un gran ejemplo de esto es la creación de compuertas lógicas para el primer proyecto de Algoritmos y Estructuras de datos 1, el semestre 2 del 2019, ya que cada compuerta era distinta era creada de un molde diferente, pero a la hora de utilizarlo no era necesario saber específicamente cómo validar las entradas.

B. Adapter

I. Propósito

Convertir la interfaz de una clase en la interfaz que espera otra clase, la del cliente. El adapter permite el trabajo en conjunto de dos clases que de otra forma no podrían trabajar entre ellas debido a la incompatibilidad de interfaces. [3]

II. Diagrama de clases

Fig. 2. Diagrama de clases del patrón Adapter utilizado como ejemplo programado

III. Estructura

Según se aprecia en el diagrama UML se cuenta con tres clases que permiten la implementación del patrón Adapter. Se tiene que el cliente solicita la reproducción de una canción en formato mp3, sin embargo, solo se cuenta con un sistema capaz de leer wav, por tanto la clase MP3player envía la canción la clase adapterMP3toWAV y esta se encarga de realizar los ajustes de formato y especificación de características (parámetros) que requiere la clase WAVplayer para funcionar y reproducir sonido, adaptando así la clase con la que interactúa el cliente con la clase que realiza la acción solicitada.

IV. Contexto

1. Al trabajar con distintos formatos de archivos dentro de un mismo sistema, por ejemplo, si se trabaja en un mismo sistema con archivos XML y JSON estos se leen y modifican de manera distinta al otro, por tanto, se debe utilizar un adapter que permita el trabajo en conjunto de ambos formatos sin ocasionar errores por diferencia de formatos.
2. Sea que se requiera una funcionalidad específica de un sistema o módulo antiguo en una nueva implementación donde estos ya no puedan interactuar directamente debido a incompatibilidad de funcionamiento ya sea por cantidad de parámetros, o dependencias se requerirá un adapter que permita la correcta comunicación entre ambos para poder utilizar la característica del modelo antiguo en la implementación actual.

Referencias bibliográficas

[1] D. NIKOLAEVA and V. BOZHIKOVA, "One Approach to Improve the Software Quality by Applying Software Design Patterns," 2019 16th Conference on Electrical Machines, Drives and Power Systems (ELMA), Varna, Bulgaria, 2019, pp. 1-6.   
Disponible: https://ieeexplore-ieee-org.ezproxy.itcr.ac.cr/document/8771691

[2] F. Khomh and Y. Guéhéneuc, "Design patterns impact on software quality: Where are the theories?," 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), Campobasso, 2018, pp. 15-25  
Disponible: https://ieeexplore-ieee-org.ezproxy.itcr.ac.cr/document/8330193

[3] GeeksforGeeks. n.d. *Adapter Pattern - Geeksforgeeks*. [online]

Disponible:<https://ww.geeksforgeeks.org/dapter-pattern/> [Accessed 10 March 2020].