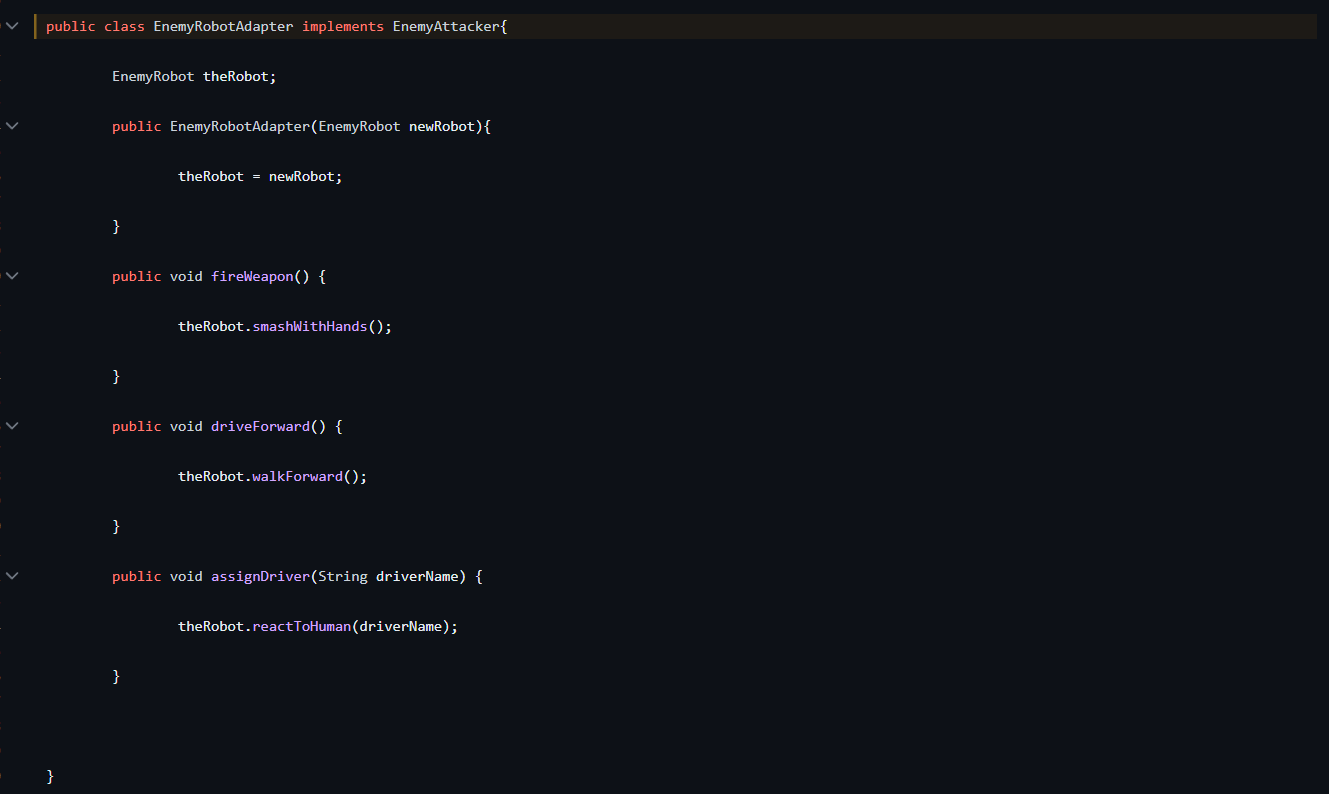
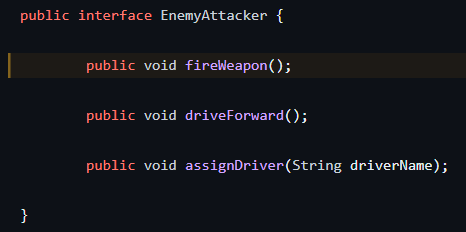
* Información y estructura del patrón en el código seleccionado.

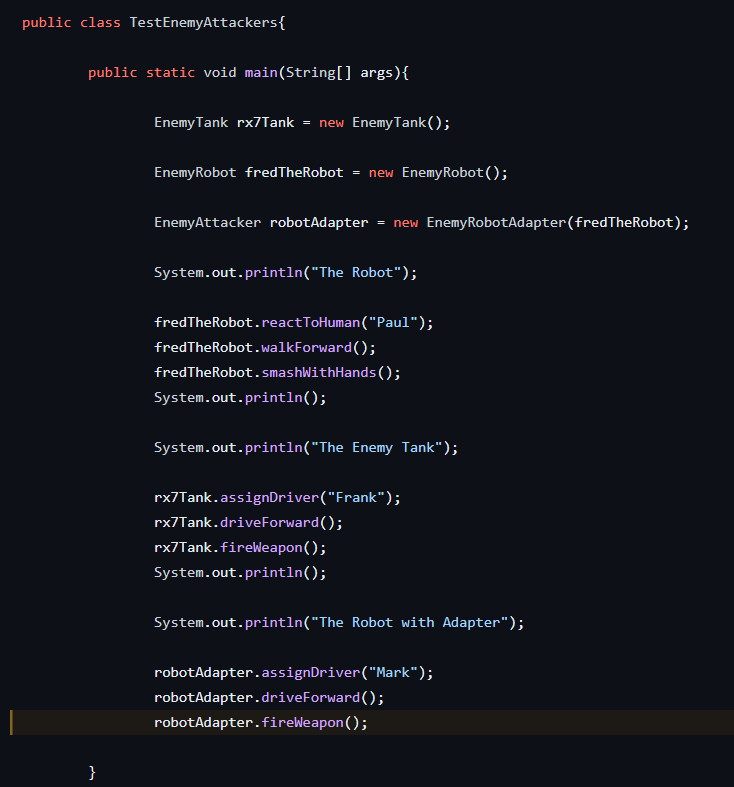
Este fragmento del código es el adaptador implementado para todas las clases que deseen implementar la clase EnemyAttavker.



Esta es la clase de la cual hereda el método EnemyRobotAdapter.



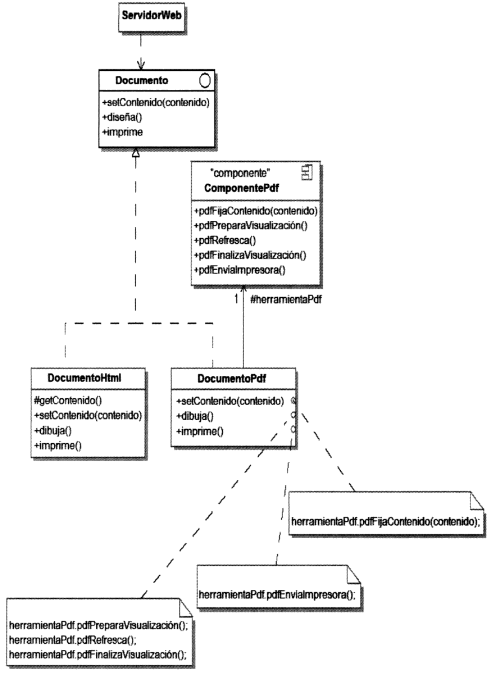
Esta es una de las clases que se ve beneficiada del uso del EnemyRobotAdapter.



* Información del patrón Adapter: este patrón permite convertir la interfaz de una clase con el fin de que esta sea compatible con otra. Es transformar una clase existente a una nueva interfaz con la cual se pueda trabajar.

Figura 1

El patrón Adapter aplicado a un componente de documentos PDF



(Debrauwer, 2012)

Este es el diagrama UML de un programa que implementa el patrón adaptar en una pagina web donde el cliente ingresa sus archivos en formato PDF y la pagina se lo muestra a el en formato HTML.

* ¿Por qué tiene sentido haber utilizado el patrón en ese punto del proyecto? ¿Qué ventajas tiene?

Es ventajoso la implementación de este patrón en el proyecto mostrado ya que permite la interacción de dos interfases que en primer lugar no son compatibles, pero mediante la implementación del adaptador. Adicionalmente tiene sentido ya que muchas veces la interfaz de entrada no siempre es la misma de salida como lo es en el ejemplo anterior donde el usuario ingresa la interfaz en forma de PDF, pero a este en su interfaz final se le muestra en HTML.

* ¿Qué desventajas tiene haber utilizado el patrón en ese punto del proyecto?

Puede presentar problemas a futuro con el mantenimiento o actualización del código ya que este patrón puede presentar problemas de compatibilidad entre las interfaces. Adicionalmente este patrón de programación aumenta la complejidad del programa ya que agrega nuevas clases a este y genera una dependencia de estas.

* ¿De qué otras formas se le ocurre que se podrían haber solucionado, en este caso particular, los problemas que resuelve el patrón?

Se podría eliminar la diferencia de interfaces ya sea en la de entrada o la de salida evitando que sea necesario la implementación del adaptador. Otra forma de solucionar el problema presentado sin la necesidad de implementar este patrón es evitar la dependencia de las dos interfaces buscando la resolución de los problemas sin necesidad de que las dos interfaces interaccionen entre ellas.

* URL: https://github.com/rick2785/JavaCode.git

# Bibliografía

Debrauwer, L. (2012). *Patrones de diseño para C#: Los 23 modelos de diseño: descripción y soluciones ilustradas en UML 2 y C#.* Ediciones ENI.