- ¿Cuáles son las estrategias?

  Las estrategias son los diferentes tipos de ataque representados por AtaqueArco, AtaqueCuchillo y AtaqueLanza. Las tres implementan la interfaz Ataque como denominador común del que heredan métodos obligatorios. Es un ejemplo del patrón de estrategia llamado patrón Strategy.

- ¿Está el código de las diferentes ataques mezclado?

  No

- ¿El código del ataque con espada es independiente del código del ataque con arco?

  Sí

- .Relaciona esto con algún principio SOLID.

  - Responsabilidad Única de SOLID: Una clase debe de tener sólo una función(responsabilidad). En este escenario, cada clase de ataque se encarga exclusivamente de definir cómo se ejecuta ese tipo de ataque.

  - Principio de segregación de interfaces: Este principio nos recuerda que ninguna clase debería depender de métodos que no usa.
    En este caso, todos los métodos de la interfaz Ataque son usados por sus clases implementadas.

  - El Principio de Inversión de Dependencias: Los módulos de alto nivel no deberían depender de los módulos de bajo nivel.
    En este caso, las clases AtaqueEspada, AtaqueArco y AtaqueCuchillo son detalles que implementan la abstracción Ataque. Los personajes, que serían los módulos de alto nivel, no dependen directamente de AtaqueEspada, AtaqueArco o AtaqueCuchillo, sino que dependen de la abstracción Ataque.

  - Principio de Sustitución de Liskov: los objetos de una clase base deben poder ser sustituidos por objetos de una clase derivada sin alterar el funcionamiento del programa.

- ¿Para que vale la interface Ataque?

  Sirve como una abstracción para diferentes estrategias de ataque. Define un conjunto de métodos que deben ser implementados por cualquier clase que represente una estrategia de ataque. Los métodos definidos en la interfaz Ataque incluyen lanzaAtaque, getNombre, getValorAtaque, crearAtaque,

- ¿Qué clase o clase se encarga del papel de contexto?

  La clase Personaje puede ser la clase de contexto, ya que parece ser la que utiliza las diferentes estrategias de ataque (AtaqueEspada, AtaqueArco, AtaqueCuchillo).

- ¿En que se parece y en que se diferencia el contexto de tu proyecto al contexto del Patrón teórico del boletín?

  Esta clase Personaje puede cambiar su comportamiento en tiempo de ejecución al cambiar la estrategia de ataque que está utilizando, que se hace a través de un método como setAtaque(Ataque ataque).

  En el patrón teórico del boletín, el contexto es el soldado que puede portar y utilizar varias armas distintas. El soldado puede cambiar su comportamiento en tiempo de ejecución al cambiar el arma que está utilizando, lo cual se hace a través de un método como cambiarArma(Arma arma).

  Las similitudes entre ambos contextos son:

  Ambos contextos utilizan diferentes estrategias (ataques o armas) que pueden ser cambiadas en tiempo de ejecución.

  Ambos contextos tienen un método para cambiar la estrategia que están utilizando.

  Las diferencias entre ambos contextos son:

  En mi proyecto, las estrategias son ataques (AtaqueEspada, AtaqueArco, AtaqueCuchillo), mientras que en el patrón Strategy, las estrategias son armas.

  En mi proyecto, el método para cambiar la estrategia es setAtaque(Ataque ataque), mientras que en el patrón teórico del boletín, el método para cambiar la estrategia es cambiarArma(Arma arma)

- ¿Quién es el Cliente?

  La clase Personaje

- ¿Como cambia el cliente de estrategia?

  Método setAtaque(Ataque ataque). Este método permite establecer una nueva estrategia de ataque para el personaje

- ¿Para cambiar de estrategia, tiene que conocer el cliente detalles de implementación de la estrategia?

  No, el cliente no necesita conocer los detalles de implementación de la estrategia para cambiarla. En mi proyecto, la clase Personaje (el cliente) cambia la estrategia de ataque utilizando el método setAtaque(Ataque ataque). Este método acepta un objeto de tipo Ataque, que es una interfaz.

- ¿Si creamos un nuevo tipo de ataque para un nuevo cliente, por ejemplo App2, es necesario modificar el cliente antiguo? Relaciona esto con algún principio SOLID.

  En principio no se necesita modificar al cliente antiguo. El patrón Strategy permite que las estrategias sean intercambiables en tiempo de ejecución.

  Está relacionado con el principio Opem/Closed de SOLID (clases abiertas zpara la extensión y cerradas para la modificación)