Refactoring

Ejercicio 2 - Refactoring

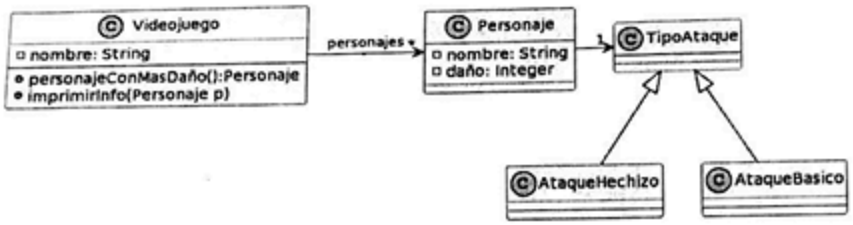
Para el siguiente código, realice las siguientes tareas:

(i) indique qué mal olor presenta

(ii) indique el refactoring que lo corrige

(iii) aplique el refactoring mostrando únicamente el código que cambió, detallando cada paso intermedio.

Si vuelve a encontrar un mal olor, retorne al paso (i).




```
public class Videojuego{
    // ...
    //
public Personaje personajeConMasDaño() {
      Personaje temp = null;
      double max= 0;
      for (Personaje p : personajes) {
            double daño = p.getTipoAtaque().calcularDaño(p.getDaño());
            if (daño > max){
                  temp = p;
                  max = daño;
           }
       }
       return temp;
 }

 public void imprimirInfo(Personaje p){
       System.out.println(p.getNombre() + "tiene como daño " + p.getDaño());
       if (p.getTipoAtaque().getClass() == AtaqueHechizo.class) {
             System.out.println("Ataque tipo hechizo");
             System.out.println("Este ataque dobla tu fuerza");
       } else{
             System.out.println("Ataque tipo Ataque Básico");
             System.out.println("Este ataque mantiene tu fuerza");
             }
 }
```

Lo resuelvo como se resuelve este año (2025). Primero detallando todos los Code Smell y luego haciendo todos los refactoring.

Code Smells:

(1) Code Smell: Nombre poco autoexplicativo:

- En el metodo personajeConMasDaño() de la clase VideoJuego, la variable Personaje temp
  (2) Code Smell: Reinventa la rueda
- El bloque for que esta dentro del metodo personajeConMasDaño()
  (3) Code Smell: Switch Statement

- En el metodo imprimirInfo(Personaje p)
  (4) Code Smell: Feature Envy
- En el metodo personajeConMasDanho(), en la linea donde se calcula el danho del personaje, el encargado de devolver el danho del ataque es el personaje.

Refactorings:

(1) Code Smell: Nombre poco autoexplicativo
Aplicamos Rename Temporary Field

- Modificamos el nombre de la variable 'temp' a 'personajeMaxDanho'
- Modificamos la referencias donde se usaba 'temp' y ahora referenciamos a 'personajeMaxDanho'

(2) Code Smell: Reinventa la rueda
Aplicamos Replace loop with pipeline

- Se debe usar los streams que son funciones que nos proporciona java.
- Ademas, en este paso podemos hacer Remove Temporary Field
  - Aplicamos Remove Temporary Field, por que la variable que le aplicamos 'Rename Temporary Field' ya no es necesaria, sino que retornamos directamente el personaje con max danho, utilizando streams.

(3) Code Smell: Switch Statement
Aplicamos Replace Conditional With Polymorphis

- Creamos una clase abstracta TipoAtaque
- Creamos por cada rama del condicional, una clase concreta que extienda de 'TipoAtaque'
  - Creamos la clase AtaqueHechizo que extiende de 'TipoAtaque'
  - Creamos la clase AtaqueBasico que extiende de 'TipoAtaque'

En la clase VideoJuego:

- Aplicamos extract method para cada rama del condicional if del metodo imprimirInfo
  - Creamos el metodo 'imprimirInfoAtaqueHechizo () '
    - Copiamos la logica de imprimir la informacion del ataque hechizo, dentro del metodo ' imprimirInfoAtaqueHechizo () '
    - Reemplazamos la logica dentro del metodo imprimirInfo, por una invocacion al metodo 'imprimirInfoAtaqueHechizo ()'
  - Creamos el metood 'imprimirInfoAtaqueBasico()'
    - Copiamos la logica de imprimir la informacion del ataque basico, dentro del metodo 'imprimirInfoAtaqueBasico ()'

        - Reemplazamos la logica dentro del metodo imprimirInfo, por una invocacion al metodo imprimirInfoAtaqueBasico ()
- Aplicamos Move Method para cada rama del condicional
    - Movemos el metodo imprimirInfoAtaqueBasico() de la clase VideoJuego a la clase AtaqueBasico
    - Movemos el metodo imprimirInfoAtaqueHechizo() de la clase VideoJuego a la clase AtaqueHechizo
- Como ambas subclases tienen el mismo comportamiento, podriamos hacer un pull up para hacer que extiendan el comportamiento de la clase TipoAtaque, pero antes, los metodos deben tener firmas compatibles, entonces:
    - Aplicamos Rename Method en el metodo imprimirInfoAtaqueHechizo de la clase AraqueHechizo
        - Cambiamos el nombre a imprimirInfo ()
    - Aplicamos Rename Method en el metodo imprimirInfoAtaqueBasico de la clase AtaqueBasico
        - Cambiamos el nombre a imprimirInfo ()

Recien ahora podemos hacer un Pull Up Method. Entonces, aplicamos Pull Up Method, y el metodo imprimirInfo() pasaria a la superclase.

- Una vez que tenemos la firma del metodo public void imprimirInfo() en la superclase (clase TipoAtaque)
    - Debemos crear dos metodos abstractos en la superclase:
        - getInformacionAtaque()
        - getInformacionTipo()
    - Modificamos el metodo imprimirInfo() para usar polimorfismo y extraemos la logica en comun.
        - Para el primer systemout extraemos la parte de "Ataque tipo" + this.getInformacionTipo();
        - Para el segundo systemout extraemos la parte de "Este ataque"+ this.getInformacionAtaque()+ " tu fuerza"

(4) Code Smell: Feature Envy

- Aplicamos Extract Method
    - Creamos el metodo getDanhoAtaque() dentro de la clase Videojuego
    - Extraemos la logica de p.getTipoAtaque().calcularDanho().getDanho() y la copiamos en el metodo creado.
    - Reemplazamos esa asignacion por una invocacion a ese metodo
- Aplicamos Move Method

- Como el personaje debe saber responder el danho de su ataque, movemos el metodo que creamos en el paso anterior 'getDanhoAtaque' a la clase Personaje
    - Modificamos la logica para utilizar polimorfismo.
        - Como cada tipo de ataque debe saber responder al metodo getDanho(), lo que hacemos es:
            - Crear un metodo abstracto en la superclase 'TipoAtaque' con firma public abstract double getDanho()
                - Hacer que las subclases redefinan esos metodos y esten obligados a retornar un danho.