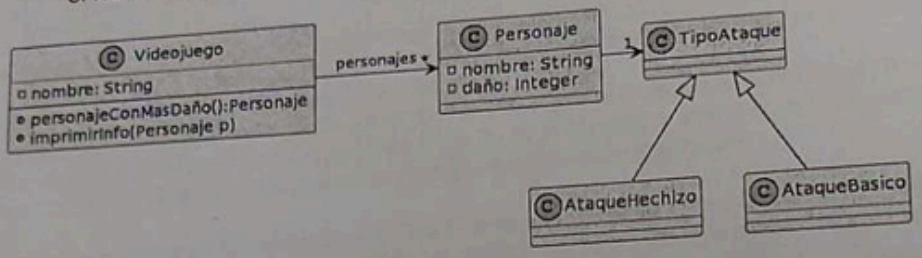
Ejercicio refactoring parcial

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");
          }
}
```

Malos olores encontrados:

- Reinventar la rueda - lineas 8-15
- Instante Of - lineas 22-28
- Feature envy - linea 21
- Null innecesario - linea 5
- Variable temporaria innecesaria - linea 6, linea 9

Refactoring - Replace loop with pipeline, eliminar variables

- Genero un algoritmo que replique el algoritmo de for pero con streams().
- Reemplazo el for loop por el pipeline recientemente creado.
- Creo una nueva variable llamada máximo de tipo Personaje el cual va a ser igual al algoritmo creado

Algoritmo hecho:

```
Personaje maximo = personajes.stream()
                .max(Comparator.comparing(personaje → personaje.getTipoAta
                .orElse(null)
```

- Elimino las variables temporales temp y max y sus llamados.
- Cambio el retorno a la nueva variable asignada (máximo).

Refactoring - Replace null with null object

El code smell ahora se encuentra dentro del algoritmo nuevo creado, el cual retorna null en caso de no encontrar un máximo.

Pasos:

Creo una clase llamada PersonajeNulo → public class PersonajeNulo.

Copio las variables, métodos y constructor de Personaje, haciendo que Personaje nulo inicialice en 0 el daño, nombre = “ ”, tipoAtaque = null y los métodos nuevos agregados que retornen 0.

Code smell, null object en tipoAtaque por parte de PersonajeNulo

Refactoring a aplicar → Replace null with Null object en tipoAtaque

Creo una nueva clase llamada AtaqueNulo que implementa TipoAtaque. Implemento los métodos necesarios por parte de TipoAtaque y retorno vacio en sus respectivos métodos (no muestra que métodos implementa tipoAtaque).

- Cambio la asignación de su constructor por un new AtaqueNulo()

Code smell encontrado, 2 clases comparten las mismas características y métodos (Personaje, PersonajeNulo)

Refactoring a aplicar → Pull up method

- Creo una clase abstracta llamada TipoPersonaje.
- Personaje y PersonajeNulo asigno que extiendan de esa clase nueva
- Subo las v.i de Personaje y elimino las v.i de PersonajeNulo, ahora van a utilizar las v.i de la clase padre (nombre, daño, tipoAtaque).
- Genero un constructor pasandole los parametros requeridos para inicializar las nuevas variables en la clase padre.

- Cambio las asignaciones de los constructores de Personaje y PersonajeNulo para que se asignen a la clase superior (super(nombre, daño, tipoAtaque).
- Subo el metodo getTipoAtaque() a la clase superior (esto para PersonajeNulo y Personaje), lo mismo con los metodos getNombre() y getDaño().
- En la clase Videojuego, el método `public Personaje personajeConMasDaño()` ahora pasa a ser `public TipoPersonaje personajeConMasDaño()`
- en `imprimirInfo(Personaje P)` , cambio el pasaje por parametro por un TipoPersonaje: `imprimirInfo(TipoPersonaje P)`

💡 **Para el code smell de Instante Of...**

Refactoring a aplicar → Replace conditional with polymorfism

- Agrego un nuevo metodo llamado imprimirInfoAtaque() a la interfaz TipoAtaque, para que las subclases deban implementar dicho método.
- Creo un método llamado imprimirInfoAtaque(TipoPersonaje p) en la clase Videojuego.
- Muevo la implementacion del primer if al nuevo metodo.
- Muevo el metodo a la clase AtaqueHechizo.
- Elimino el parametro pasado en el metodo.
- Creo el mismo método pero en este caso para la implementacion del else.
- Muevo el método a la clase AtaqueBasico y elimino el parametro pasado.
- Creo un tercer método con el mismo nombre.
- Implemento lo necesario para que imprima vacio.
- Muevo el método a la clase AtaqueNulo y elimino el parámetro pasado.

- Elimino el if y else del método de imprimirInfo en Videojuego, reemplazándolo por la llamada al método imprimirInfoAtaque() del TipoAtaque que posea pasado por parámetro.

Code smell de la linea 21:

Refactoring a aplicar → Move Method

- Creo un método en la clase Videojuego llamado `imprimirInfoPersonaje(TipoPersonaje p)`
- muevo el bloque de código de la linea 21 al método creado.
- Aplico Move Method y muevo el Metodo a la clase abstracta de TipoPersonaje.
- Elimino el parámetro pasado en el método.
- Cambio las llamadas a p por las llamadas de la propia clase.
- Reemplazo el bloque de codigo con code smell por una llamada al metodo recientemente movido (p.imprimirInfoPersonaje).

Código refactorizado:

```
public class Videojuego {
    // ...
    //
    public TipoPersonaje personajeConMasDaño() {
        Personaje maximo = personajes.stream()
                        .max(Comparator.comparing(personaje → personaje.getTipoAta
                        .orElse(null)

        return maximo;
    }
```

```
    public void imprimirInfo(TipoPersonaje p) {

        p.imprimirInfoAtaque();
    }

}


public abstract class TipoPersonaje {
    private String nombre;
    private int daño;
    private TipoAtaque tipoAtaque;

    public TipoPersonaje(String nombre, int daño TipoAtaque tipoAtaque) {
        this.nombre = nombre;
        this.daño = daño;
        this.tipoAtaque = tipoAtaque;
    }

    public getTipoAtaque() {
        return tipoAtaque;
    }


    public void imprimirInfoPersonaje() {
        System.out.println(getNombre() + "tiene como danio " + getDanio());
    }


}

public class PersonajeNulo extends TipoPersonaje{
    Public PersonajeNulo() {
        super("", 0, new AtaqueNulo();
    }
```

```
}


public class Personaje extends TipoPersonaje {
   Public Personaje(String nombre, int daño, TipoAtaque tipoAtaque) {
      super(nombre, daño, tipoAtaque
   }

}



public interface TipoAtaque {
   public void imprimirInfoAtaque();
}

public class AtaqueHechizo implements TipoAtaque {

   public void imprimirInfoAtaque() {
         System.out.println("Ataque tipo hechizo");
         System.out.println("Este ataque dobla tu fuerza");
   }
}

public class AtaqueBasico implements TipoAtaque {

   public void imprimirInfoAtaque() {
      System.out.println("Ataque tipo Ataque Basico");
      System.out.println("Este ataque matniene tu fuerza");
   }

}


public class AtaqueNulo implements TipoAtaque {
   public void imprimirInfoAtaque() {
```

```
        System.out.println("vacio");
    }
}
```