**Diseño de Software**

TAREA 2 SOLID

**Integrantes Grupo 2:**

Bryan Jair Segovia Mariscal

Kenny Gabriel Jara

**Repositorios separados:**

<https://github.com/narcizo/Pokemon>

<https://github.com/reymundotenorio/ConsultorioMedico.git>

**Índice**

Principio de OCP ------------------------------------------------------------------ pag 3

Principio de OCP e Interface Segregation ----------------------------------- pag 4

Principio de Dependency Inversión Principle(DIP) ------------------------- pag 5

Principio de Single Responsability --------------------------------------------- pag 6

Interfaz de usuario gráfica, Texto, Aplicación, Word

Descripción generada automáticamente**1) Principio de OCP**

La clase “Tipo” es un emum, esto viola el principio de “Open-Closed”, ya que no permitiría agregar más tipos de pokémones.

Captura de pantalla de computadora

Descripción generada automáticamente**Propuesta de solución.**

Crear una clase llamada “Tipo”, que permita la creación de los tipos de Ataque.

**2) Principio de OCP e Interface Segregation**

Captura de pantalla de computadora

Descripción generada automáticamente

Todas estas clases están dentro del paquete “Ataque”, cada una de ellas posee el método “efeito()”.

Captura de pantalla de computadora

Descripción generada automáticamente**Propuesta de solución**

Crearemos una interface llamada “Atack”, ella tendrá el método “efeito()”, y las clases presentadas anteriormente implementarán la interface y el método.

**3) Dependency Inversión Principle(DIP)**

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

La mayoría de las clases del Proyecto cuentan con estos atributos, eso viola el principio DIP.

**Propuesta de solución.**

Captura de pantalla de computadora

Descripción generada automáticamentePodríamos crear una clase a la que llamaremos “Comun”, y en ella pondremos esos atributos, y haremos que las clases que usen esos atributos extiendan de esta.

**Repositorio dos**

**Single Responsability**

En la clase paciente se puede observar que existen métodos que son repetitivos y que realmente no deberían estar en esa clase. En esta clase paciente deberían estar los datos del paciente nada mas con sus getters y setters, pero se ve que hay métodos como agregar paciente, actualizar paciente, activar o desactivar paciente, entonces estos métodos deberían ir en otra clase que maneje los distintos cambios que se le hace al paciente.

Esto también lo podemos evidenciar en la clase medico, por lo que sería útil tener otra clase que realice los métodos tanto AGREGAR, ACTIVAR y DESACTIVAR medico

Así:

Mecico{

public static ResultSet resultado;

|  |
| --- |
| public static void Usuario\_Medico(int ID\_M, int ID\_U, String Usuario){ |
|  |  |
|  |  |
|  |  |
|  | try{ |
|  |  |
|  | CallableStatement consulta = Conexion.con.prepareCall("{call UsuarioMedico (?,?)}"); |
|  |  |
|  | consulta.setInt(1, ID\_M); |
|  | consulta.setInt(2, ID\_U); |
|  | consulta.execute(); |
|  | JOptionPane.showMessageDialog(null,"Medico agregado con Nombre Usuario: "+Usuario, |
|  | "Información",JOptionPane.INFORMATION\_MESSAGE); |
|  |  |
|  |  |
|  |  |
|  | }catch(SQLException ex){ |
|  |  |
|  | JOptionPane.showMessageDialog(null,ex.getMessage(),"Error",JOptionPane.ERROR\_MESSAGE); |
|  |  |
|  | } |
|  | } |

}

MedicoBD{

public static void Agregar\_Medico(String Nombre, String Apellido, int ID\_Especialidad){

…

}

public static void Actualizar\_Medico(int ID, String Nombre, String Apellido, int ID\_Especialidad){

…

}

public static void Usuario\_Medico(int ID\_M, int ID\_U, String Usuario){

…

}

}

**Open Closed**

Existen muchos usuarios que tienen los mismos métodos como el de agregar, actualizar, activar o desactivar entonces aquí se ve claramente que todos estos usuarios como el paciente, el medico deberían ser una subclase de usuario. Donde el usuario ya tengo estos métodos en su clase entonces ya no se tendría que modificar los métodos si se quiere agregar otro participante en el sistema.