Implementación del patrón Composite en el sistema de Citas Medicas

1.Introducción

Composite es un patrón de diseño estructural que te permite componer objetos en estructuras de árbol y trabajar con esas estructuras como si fueran objetos individuales.

2.¿Qué es el patrón composite?

El concepto básico del patrón Composite consiste en representar objetos simples y sus containers (o contenedores, también llamados colecciones en algunos lenguajes, o sea: grupos de objetos) en una clase abstracta de manera que puedan ser tratados uniformemente. Este tipo de estructura se conoce como jerarquía parte-todo (en inglés: part-whole hierarchy), en la que un objeto es siempre, o una parte de un todo, o un todo compuesto por varias partes.

3.Participantes

- CompositeView: Una vista compuesta es una vista a la que se le han agregado varias subvistas.
- ViewManager: El manejador de vista se ocupa de la inclusión de porciones de fragmentos de plantilla en la vista compuesta.
- HeaderView: Es una subvista incluida dentro de la principal.
- FooterView: Es una subvista incluida dentro de la principal.

4.**Problema**

Por ejemplo, tenemos dos tipos de objetos: Usuario y Medico. Un Médico es un Usuario, y varios usuarios conforman un grupo de usuarios, además estos, pueden ser un componente del hospital.

Este Grupo de Usuarios también pueden contener algunos Médicos o incluso Usuarios diferentes.

Al crear un sistema con estas clases podemos implementar un grupo de usuarios que contenga diferentes tipos de usuarios. Para esto primero tenemos que conocer las clases y el nivel de anidación al que podemos llegar con ellas.

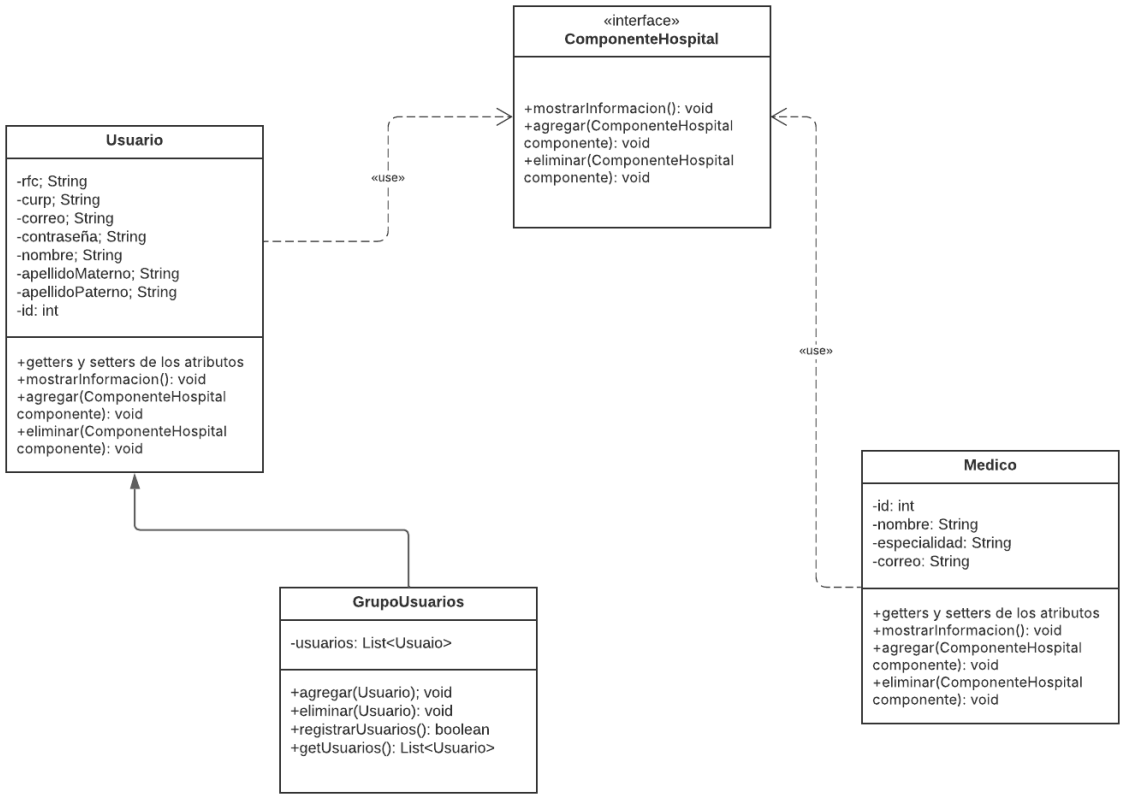
5.**Solución**

El patrón Composite trabajara con las clases Usuario y Medico a través de una interfaz común que declara un método para mostrar la información de los diferentes usuarios.

En este sentido, Usuario podría ser una superclase con diferentes tipos de usuarios (Paciente, Medico, Administrador). En este patrón también se implementará una clase llamada GrupoUsuarios para crear una lista de diferentes usuarios.

Modificanso estas clases, tenemos que modificar las vista de inicio de sesión para poder agregar un solo usuario o agregar un grupo de ellos.

6.Diagrama UML



7.Código de implementación

- Inteface ComponenteHospital

```
public interface ComponenteHospital {
   void mostrarInformacion();
   void agregar(ComponenteHospital componente); // Solo para composites
   void eliminar(ComponenteHospital componente); // Solo para composites
   List<ComponenteHospital> getSubordinados(); // Solo para composites
}
```

- Clase Usuario

```
public class Usuario implements ComponenteHospital {
   private String  rfc, curp, correo, contraseña, nombre, apellidoMaterno, apellidoPaterno, tipo;
   private int id;

   public Usuario(String rfc, String curp, String correo, String contraseña, String nombre, String
apellidoMaterno, String apellidoPaterno) {
      this.rfc = rfc;
      this.curp = curp;
      this.correo = correo;
      this.contraseña = contraseña;
      this.nombre = nombre;
      this.apellidoMaterno = apellidoMaterno;
      this.apellidoPaterno = apellidoPaterno;
   }

   public int getId() {
      return id;
   }

   public void setId(int id) {
      this.id = id;
   }

   public void setTipo(String tipo) {
      this.tipo = tipo;
   }

   public String getTipo() {
      return tipo;
   }

   public String getRfc() {
      return rfc;
   }

   public String getCurp() {
      return curp;
   }

   public String getCorreo() {
      return correo;
   }
```

```
public String getContraseña() {
   return contraseña;
}

public String getNombre() {
   return nombre;
}

public String getApellidoMaterno() {
   return apellidoMaterno;
}

public String getApellidoPaterno() {
   return apellidoPaterno;
}

public void setRfc(String rfc) {
   this.rfc = rfc;
}

public void setCurp(String curp) {
   this.curp = curp;
}

public void setCorreo(String correo) {
   this.correo = correo;
}

public void setContraseña(String contraseña) {
   this.contraseña = contraseña;
}

public void setNombre(String nombre) {
   this.nombre = nombre;
}

public void setApellidoMaterno(String apellidoMaterno) {
   this.apellidoMaterno = apellidoMaterno;
}

public void setApellidoPaterno(String apellidoPaterno) {
   this.apellidoPaterno = apellidoPaterno;
}
```

```
    @Override
    public void mostrarInformacion() {
        System.out.println("Usuario: " + nombre + " " + apellidoPaterno + " - RFC: " + rfc);
    }

    @Override
    public void agregar(ComponenteHospital componente) {
        throw new UnsupportedOperationException("No aplicable para usuarios individuales");
    }

    @Override
    public void eliminar(ComponenteHospital componente) {
        throw new UnsupportedOperationException("No aplicable para usuarios individuales");
    }

    @Override
    public List<ComponenteHospital> getSubordinados() {
        return Collections.emptyList(); // No tiene subordinados
    }
}
```

- Clase Medico

```
public class Medico implements ComponenteHospital {
    private int id;
    private String nombre;
    private String especialidad;
    private String correo;
    public Medico(int id, String nombre, String especialidad, String correo) {
        this.id = id;
        this.nombre = nombre;
        this.especialidad = especialidad;
        this.correo=correo;
    }

    public int getId() {
        return id;
    }

    public String getCorreo() {
        return correo;
    }

    public void setCorreo(String correo) {
        this.correo = correo;
```

```
    }

    public String getNombre() {
        return nombre;
    }

    public String getEspecialidad() {
        return especialidad;
    }

    @Override
    public void mostrarInformacion() {
        System.out.println("Médico: " + nombre + " - Especialidad: " + especialidad);
    }

    @Override
    public void agregar(ComponenteHospital componente) {
        throw new UnsupportedOperationException("No aplicable para usuarios individuales");
    }

    @Override
    public void eliminar(ComponenteHospital componente) {
        throw new UnsupportedOperationException("No aplicable para usuarios individuales");
    }

    @Override
    public List<ComponenteHospital> getSubordinados() {
        return Collections.emptyList(); // No tiene subordinados
    }
}
```

- Clase GrupoUsuarios

```
public class GrupoUsuarios {
    private List<Usuario> usuarios;
    private Connection conn; // Conexión a la base de datos

    // Constructor
    public GrupoUsuarios(Connection conn) {
        this.usuarios = new ArrayList<>();
        this.conn = conn;
    }

    // Agregar usuario al grupo
```

```
    public void agregarUsuario(Usuario usuario) {
        if (usuario != null) {
            usuarios.add(usuario);
        }
    }

    // Eliminar usuario del grupo
    public void eliminarUsuario(Usuario usuario) {
        usuarios.remove(usuario);
    }

    // Registrar todos los usuarios del grupo
    public boolean registrarUsuarios() {
        if (usuarios.isEmpty()) {
            return false;
        }

        try {
            for (Usuario usuario : usuarios) {
                RegistrarUsuario.registrar(conn, usuario);
            }
            usuarios.clear(); // Limpiar la lista después de registrar
            return true;
        } catch (SQLException ex) {
            ex.printStackTrace();
            return false;
        }
    }

    // Obtener la lista de usuarios
    public List<Usuario> getUsuarios() {
        return usuarios;
    }

    // Verificar si el grupo está vacío
    public boolean estaVacio() {
        return usuarios.isEmpty();
    }
}
```