Implementación del Patrón Factory en el Sistema de Citas Médicas

1. Introducción

En el desarrollo de nuestro sistema de citas médicas, nos enfrentamos al desafío de manejar diferentes tipos de usuarios, como **médicos** y **pacientes**, de manera flexible y escalable. Para resolver este problema, decidimos implementar el **Patrón Factory**. Este patrón nos permite crear objetos (como médicos o pacientes) sin especificar sus clases concretas, lo que hace que el sistema sea más modular y fácil de mantener.

---

2. ¿Qué es el Patrón Factory?

El **Patrón Factory** es un patrón de diseño creacional que proporciona una interfaz para crear objetos en una superclase, pero permite que las subclases alteren el tipo de objetos que se crearán. En otras palabras, el patrón Factory delega la creación de objetos a una clase especializada (la fábrica), en lugar de crear los objetos directamente en el código cliente.

---

3. Participantes del Patrón Factory

En nuestra implementación, los participantes son:

1. **Factory (PersonaFactory)**:
   - Es la clase que contiene el método `crearPersona()`, el cual decide qué tipo de objeto crear (`Usuario` o `Medico`) basándose en un parámetro.
2. **Product (Usuario y Medico)**:
   - Son las clases que representan los objetos que se crearán. En este caso, `Usuario` representa a un paciente y `Medico` representa a un médico.
3. **Cliente (Vista_Registro_Usuario)**:

- Es la clase que utiliza la fábrica (`PersonaFactory`) para crear objetos (`Usuario` o `Medico`).

---

4. ¿Cómo Funciona el Patrón Factory?

El patrón Factory funciona de la siguiente manera:

1. El **Cliente** (en este caso, `Vista_Registro_Usuario`) solicita a la fábrica (`PersonaFactory`) que cree un objeto.
2. La fábrica decide qué tipo de objeto crear (`Usuario` o `Medico`) basándose en un parámetro.
3. La fábrica devuelve el objeto creado al cliente, que lo utiliza sin necesidad de conocer los detalles de su creación.

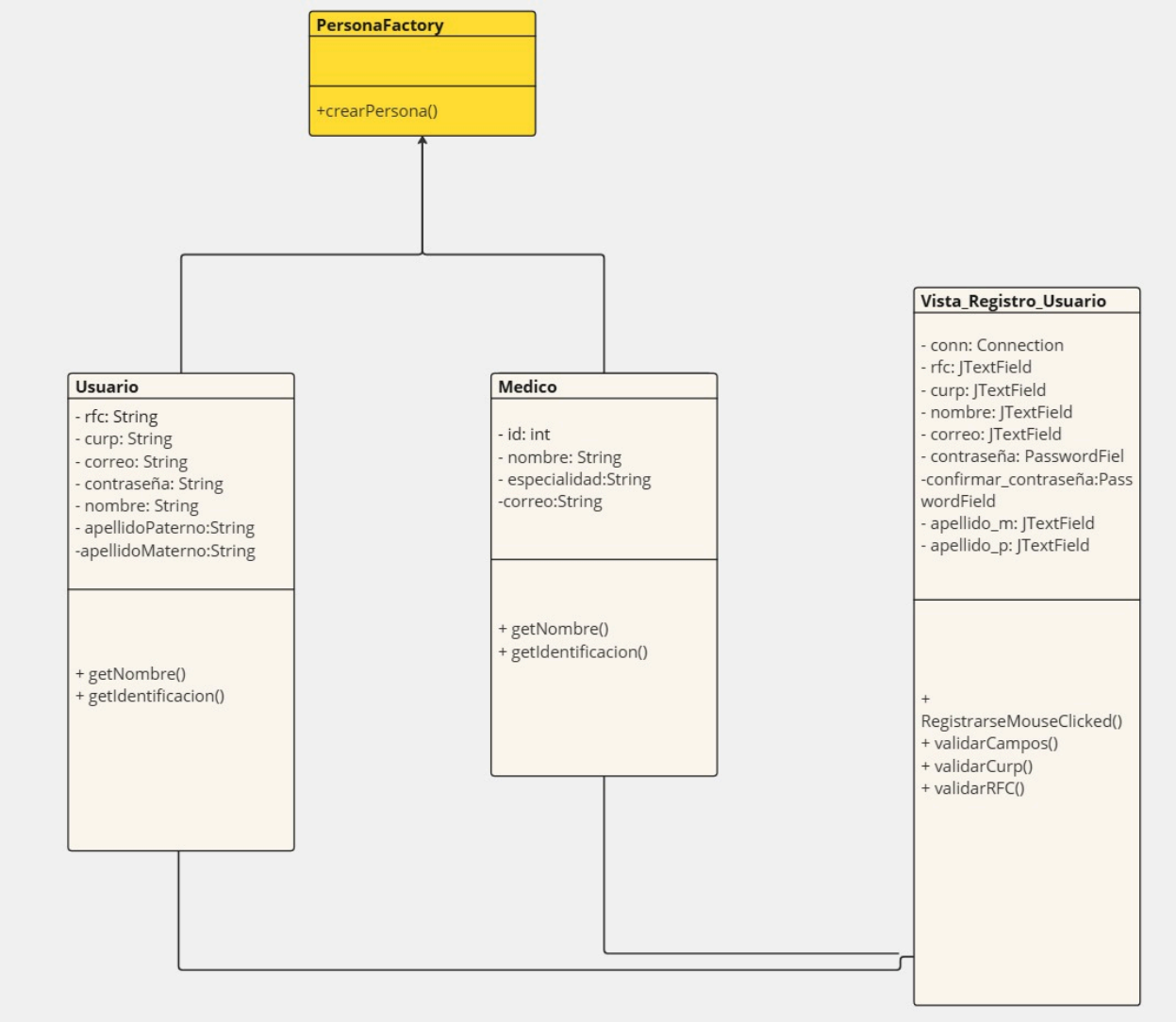
---

5. Implementación en el Proyecto

En nuestro sistema de citas médicas, implementamos el **Patrón Factory** para manejar la creación de dos tipos de usuarios: **pacientes** (representados por la clase `Usuario`) y **médicos** (representados por la clase `Medico`)

---

6. Diagrama UML



7. Código e implementación

1. Clase PersoanFactory

```
package Modelo;

import Clases.Usuario;
import Clases.Medico;

public class PersonaFactory {

    public static Object crearPersona(String tipo, String[] datos) {
```

```
        switch (tipo.toLowerCase()) {
            case "usuario":
                return new Usuario(datos[0], datos[1], datos[2], datos[3], datos[4], dat
            case "medico":
                return new Medico(Integer.parseInt(datos[0]), datos[1], datos[2], datos
            default:
                throw new IllegalArgumentException("Tipo de persona no válido: " + tip
        }
    }
}
```

2. Clase Usuario

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit th
 */
package Clases;

/**
 *
 * @author crist
 */
public class Usuario {

    private String rfc, curp, correo, contraseña, nombre, apellidoMaterno, apellido
    private int id;

    public int getId() {
        return id;
    }

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

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

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

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;
    }

}
```

3. Clase Médico

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit th
 */
package Clases;

public class Medico {

    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 String toString() {
        return nombre + " (" + especialidad + ")";
    }
}
```

4. Implementación en los botones mouseClicked

```
private void RegistrarseMouseClicked(java.awt.event.MouseEvent evt) {//GEN-F
        // TODO add your handling code here:
        String rfcText = rfc.getText();
        String curpText = curp.getText();
        String Nombre = nombre.getText();
        String correoText = correo.getText();
        String contraseñaText = new String(contraseña.getPassword());
        String confirmarContraseñaText = new String(confirmar_contraseña.getPass
        String apellidoMaternoText = apellido_m.getText();
        String apellidoPaternoText = apellido_p.getText();

        if (validarCampos(Nombre, correoText, contraseñaText, confirmarContraseñ
            if (validarCurp(curpText) && validarRFC(rfcText)) {
                if (!contraseñaText.equals(confirmarContraseñaText)) {
                    JOptionPane.showMessageDialog(this, "Las contraseñas no coincide
```

```
            return;
        }

        // Crear el objeto Usuario usando la fábrica
        String[] datosUsuario = {rfcText, curpText, correoText, contraseñaText,
        Usuario usuario = (Usuario) PersonaFactory.crearPersona("usuario", da

        try {
            if (RegistrarUsuario.registrar(conn, usuario)) {
                Iniciar_Sesion iniciar = new Iniciar_Sesion(conn);
                iniciar.setVisible(true);
                this.setVisible(false);
            }
        } catch (SQLException ex) {
            Logger.getLogger(Vista_Registro_Usuario.class.getName()).log(Leve
        }
    }
}
```

```
private void confirmar_contraseñaActionPerformed(java.awt.event.ActionEvent
    String rfcText = rfc.getText();
    String curpText = curp.getText();
    String Nombre = nombre.getText();
    String correoText = correo.getText();
    String contraseñaText = new String(contraseña.getPassword());
    String confirmarContraseñaText = new String(confirmar_contraseña.getPas
    String apellidoMaternoText = apellido_m.getText();
    String apellidoPaternoText = apellido_p.getText();
    if (Nombre.length() == 0 || correoText.length() == 0 || contraseñaText.length
        JOptionPane.showMessageDialog(this, "Rellena todos los campos", "Erro
        return;
    }
```

```
        if (validarCurp(curpText) && validarRFC(rfcText)) {

            if (!contraseñaText.equals(confirmarContraseñaText)) {
                JOptionPane.showMessageDialog(this, "Las contraseñas no coinciden"
                return;
            }

            // Crear el objeto Usuario si la contraseña coincide
            String[] datosUsuario = {rfcText, curpText, correoText, contraseñaText, No
            Usuario usuario = (Usuario) PersonaFactory.crearPersona("usuario", dato

            try {
                if (RegistrarUsuario.registrar(conn, usuario)) {
                    Iniciar_Sesion iniciar = new Iniciar_Sesion(conn);
                    iniciar.setVisible(true);
                    this.setVisible(false);
                }
            } catch (SQLException ex) {
                Logger.getLogger(Vista_Registro_Usuario.class.getName()).log(Level.S
            }
        }
    }//GEN-LAST:event_con
```

8. Conclusión

El Patrón Factory ha sido una excelente elección para nuestro sistema de citas médicas, ya que nos permite manejar la creación de diferentes tipos de usuarios de manera modular y escalable. Gracias a este patrón, el sistema es más fácil de mantener y extender en el futuro.