	Computación	Docente: Diego Quisi Peralta
	Programación Aplicada	Período Lectivo: Septiembre 2020 – Febrero 2021

		FORMATO DE GUÍA DE PRÁCTICA DE LABORATORIO / TALLERES / CENTROS DE SIMULACIÓN – PARA DOCENTES	
CARRERA: COMPUTACIÓN/INGENIERÍA DE SISTEMAS		ASIGNATURA: PROGRAMACIÓN APLICADA	
NRO. PROYECTO:	1.1	TÍTULO PROYECTO: Prueba Practica 1 Desarrollo e implementación de un sistema de gestión de matrimonios de la ciudad de Cuenca	
OBJETIVO: Reforzar los conocimientos adquiridos en clase sobre la programación aplicada (Java 8, Programación Genérica, Reflexión y Patrones de Diseño) en un contexto real.			
INSTRUCCIONES:		1. Revisar el contenido teórico y práctico del tema	
		2. Profundizar los conocimientos revisando los libros guías, los enlaces contenidos en los objetos de aprendizaje Java y la documentación disponible en fuentes académicas en línea.	
		3. Deberá desarrollar un sistema informático para la gestión de matrimonios, almacenar en archivos y una interfaz gráfica.	
		4. Deberá generar un informe de la práctica en formato PDF y en conjunto con el código se debe subir al GitHub personal.	
		5. Fecha de entrega: El sistema debe ser subido al git hasta 27 de noviembre del 2020 – 23:55.	
ACTIVIDADES POR DESARROLLAR			

1. Enunciado:

Realizar el diagrama de clase y el programa para gestionar los matrimonios de la ciudad de Cuenca empleando las diferentes técnicas de programación revisadas en clase.

Problema: De cada matrimonio se almacena la fecha, el lugar de la celebración y los datos personales (nombre, apellido, cédula, dirección, genero y fecha de nacimiento) de los contrayentes. Es importante validar la equidad de genero.

Igualmente se guardar los datos personales de los dos testigos y de la autoridad civil (juez o autoridad) que formalizan el acto. Además de gestionar la seguridad a través de un sistema de Usuarios y Autentificación.

Calificación:

- Diagrama de Clase 20%
- MVC: 20%
- Patrón de Diseño aplicado: 30%
- Técnicas de Programación aplicadas (Java 8, Reflexión y Programación Genérica): 20%
- Informe: 10%

2. Informe de Actividades:


- Planteamiento y descripción del problema.
- Diagramas de Clases.
- Patrón de diseño aplicado
- Descripción de la solución y pasos seguidos.
- Conclusiones y recomendaciones.
- Resultados.

RESULTADO(S) OBTENIDO(S):

- Interpreta de forma correcta los algoritmos de programación y su aplicabilidad.
- Identifica correctamente qué herramientas de programación se pueden aplicar.

CONCLUSIONES:

- Los estudiantes identifican las principales estructuras para la creación de sistemas informáticos.
- Los estudiantes implementan soluciones gráficas en sistemas.
- Los estudiantes están en la capacidad de implementar la persistencia en archivos.

	Computación	Docente: Diego Quisi Peralta
	Programacion Aplicada	Período Lectivo: Septiembre 2020 – Febero 2021

RECOMENDACIONES:

- Revisar la información proporcionada por el docente previo a la práctica.
- Haber asistido a las sesiones de clase.
- **Consultar con el docente las dudas que puedan surgir al momento de realizar la prueba.**

BIBLIOGRAFIA:

[1]: <https://www.ups.edu.ec/evento?calendarBookingId=98892>

Docente / Técnico Docente: Ing. Diego Quisi Peralta Msc.

Firma: _____



FORMATO DE INFORME DE PRÁCTICA DE LABORATORIO / TALLERES /
CENTROS DE SIMULACIÓN – PARA ESTUDIANTES

CARRERA:

ASIGNATURA:

NRO. PRÁCTICA:

TÍTULO PRÁCTICA:

OBJETIVO ALCANZADO:

Se desarrollo una aplicación de registro de matrimonio con login, con los conocimientos previos de Programación genérica, Reflexión y Java 8

ACTIVIDADES DESARROLLADAS

1. Planteamiento y descripción del problema.

1. Enunciado:

Realizar el diagrama de clase y el programa para gestionar los matrimonios de la ciudad de Cuenca empleando las diferentes técnicas de programación revisadas en clase.

Problema: De cada matrimonio se almacena la fecha, el lugar de la celebración y los datos personales (nombre, apellido, cédula, dirección, género y fecha de nacimiento) de los contrayentes. Es importante validar la equidad de género.


Igualmente se guardar los datos personales de los dos testigos y de la autoridad civil (juez o autoridad) que formalizan el acto. Además de gestionar la seguridad a través de un sistema de Usuarios y Autentificación.

Diagrama UML

Patrón de diseño aplicado

Se aplico el patrón de diseño FactoryMethod, el cual consiste en utilizar una clase constructora abstracta con unos cuantos métodos definidos y otro abstracto: el dedicado a la construcción de objetos de un subtipo de un tipo determinado. En este caso se utilizó un método concreto validar () el cual verifica los roles de cada persona ya sea un usuario, un contrayente o una autoridad.

Descripción de la solución y pasos:

	Computación	Docente: Diego Quisi Peralta
	Programación Aplicada	Período Lectivo: Septiembre 2020 – Febrero 2021

Con la ayuda de la arquitectura MVC se organizó de una mejor manera el programa, lo primero que se hizo fue crear las clases del modelo, que son autoridad, matrimonio y persona, donde persona es la clase padre, y como usuario para el login se lo considero a autoridad.

Luego con la ayuda de los conocimientos previos de Programación genérica se utilizó el principio de reflexión en java, el cual se aplicó en el controlador, donde se creó una clase padre abstracta y se generalizó los métodos crud y aparte un método concreto para validar los tipos de objetos, por último se diseñaron las respectivas interfaces gráficas en la vista.

Modelo

Clase Persona

```
package ec.edu.ups.modelo;

import java.util.Date;
import java.util.Objects;

/**
 *
 * @author user
 */
public class Persona {

    private String cedula;
    private String nombres;
    private String apellidos;
    private String direccion;
    private String genero;
    private Date fechaNacimiento;
    private String estadoCivil;
    private String rol;

    public Persona() {
    }

    public Persona(String cedula, String nombres, String apellidos, String direccion, String genero, Date fechaNacimiento, String estadoCivil, String rol) {
        this.cedula = cedula;
        this.nombres = nombres;
        this.apellidos = apellidos;
        this.direccion = direccion;
        this.genero = genero;
        this.fechaNacimiento = fechaNacimiento;
        this.estadoCivil = estadoCivil;
        this.rol = rol;
    }

    public String getCedula() {
        return cedula;
    }

    public void setCedula(String cedula) {
        this.cedula = cedula;
    }
}
```

```
public String getNombres() {
    return nombres;
}

public void setNombres(String nombres) {
    this.nombres = nombres;
}

public String getApellidos() {
    return apellidos;
}

public void setApellidos(String apellidos) {
    this.apellidos = apellidos;
}

public String getDireccion() {
    return direccion;
}

public void setDireccion(String direccion) {
    this.direccion = direccion;
}

public String getGenero() {
    return genero;
}

public void setGenero(String genero) {
    this.genero = genero;
}

public Date getFechaNacimiento() {
    return fechaNacimiento;
}

public void setFechaNacimiento(Date fechaNacimiento) {
    this.fechaNacimiento = fechaNacimiento;
}

public String getRol() {
    return rol;
}

public void setRol(String Rol) {
    this.rol = Rol;
}

public String getEstadoCivil() {
    return estadoCivil;
}

public void setEstadoCivil(String EstadoCivil) {
    this.estadoCivil = EstadoCivil;
}

@Override
public int hashCode() {
    int hash = 7;
    hash = 83 * hash + Objects.hashCode(this.cedula);
    return hash;
}
```

```

    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Persona other = (Persona) obj;
        if (!Objects.equals(this.cedula, other.cedula)) {
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append("Persona{cedula=").append(cedula);
        sb.append(", nombres=").append(nombres);
        sb.append(", apellidos=").append(apellidos);
        sb.append(", direccion=").append(direccion);
        sb.append(", genero=").append(genero);
        sb.append(", fechaNacimiento=").append(fechaNacimiento);
        sb.append(", EstadoCivil=").append(estadoCivil);
        sb.append(", Rol=").append(rol);
        sb.append('}');
        return sb.toString();
    }
}

```

Clase Autoridad (Usuario)

```

package ec.edu.ups.modelo;

import java.util.Date;

/**
 *
 * @author user
 */
public class Autoridad extends Persona {

    private String cargo;

    private String correo;

    private String pass;

    public Autoridad(String cedula, String nombre, String apellido, String direccion, String genero,
        Date fechaNacimiento, String estadoCivil, String rol, String cargo,
        String correo, String pass) {
        super(cedula, nombre, apellido, direccion, genero, fechaNacimiento, estadoCivil, rol);
        this.cargo = cargo;
    }
}

```

```

        this.correo = correo;
        this.pass = pass;
    }

    public Autoridad() {

    }

    public String getCargo() {
        return cargo;
    }

    public void setCargo(String cargo) {
        this.cargo = cargo;
    }

    public String getCorreo() {
        return correo;
    }

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

    public String getPass() {
        return pass;
    }

    public void setPass(String pass) {
        this.pass = pass;
    }

    @Override
    public String toString() {
        return "Autoridad{" + super.getNombres() + " " + super.getApellidos() + " "
+ super.getCedula() + " " + super.getDireccion() + " " + super.getGenero() + " " +
super.getRol() + "cargo=" + cargo + ", correo=" + correo + ", pass=" + pass + '}';
    }
}

```

Clase Matrimonio

```

package ec.edu.ups.modelo;

import java.util.Date;

public class Matrimonio {

    private int codigoM;

    private String lugar;
    private Date fecha;

    private Persona contrayente1;
    private Persona contreyente2;

    private Persona testigo1;
    private Persona testigo2;
}

```



```
private Autoridad autoridad;

public Matrimonio(int codigoM, String lugar, Date fecha, Persona contrayente1,
Persona contreyente2, Persona testigo1, Persona testigo2, Autoridad autoridad) {
    this.codigoM = codigoM;
    this.lugar = lugar;
    this.fecha = fecha;
    this.contrayente1 = contrayente1;
    this.contreyente2 = contreyente2;
    this.testigo1 = testigo1;
    this.testigo2 = testigo2;
    this.autoridad = autoridad;
}

public int getCodigoM() {
    return codigoM;
}

public void setCodigoM(int codigoM) {
    this.codigoM = codigoM;
}

public Persona getContrayente1() {
    return contrayente1;
}

public void setContrayente1(Persona contrayente1) {
    this.contrayente1 = contrayente1;
}

public Persona getContreyente2() {
    return contreyente2;
}

public void setContreyente2(Persona contreyente2) {
    this.contreyente2 = contreyente2;
}

public Persona getTestigo1() {
    return testigo1;
}

public void setTestigo1(Persona testigo1) {
    this.testigo1 = testigo1;
}

public Persona getTestigo2() {
    return testigo2;
}

public void setTestigo2(Persona testigo2) {
    this.testigo2 = testigo2;
}

public Persona getAutoridad() {
    return autoridad;
}

public void setAutoridad(Autoridad autoridad) {
    this.autoridad = autoridad;
}
```

```

public String getLugar() {
    return lugar;
}

public void setLugar(String lugar) {
    this.lugar = lugar;
}

public Date getFecha() {
    return fecha;
}

public void setFecha(Date fecha) {
    this.fecha = fecha;
}

@Override
public int hashCode() {
    int hash = 7;
    hash = 37 * hash + this.codigoM;
    return hash;
}

@Override
public boolean equals(Object obj) {
    if (this == obj) {
        return true;
    }
    if (obj == null) {
        return false;
    }
    if (getClass() != obj.getClass()) {
        return false;
    }
    final Matrimonio other = (Matrimonio) obj;
    if (this.codigoM != other.codigoM) {
        return false;
    }
    return true;
}

@Override
public String toString() {
    StringBuilder sb = new StringBuilder();
    sb.append("Matrimonio{codigoM=").append(codigoM);
    sb.append(", lugar=").append(lugar);
    sb.append(", fecha=").append(fecha);
    sb.append(", contrayente1=").append(contrayente1);
    sb.append(", contrayente2=").append(contrayente2);
    sb.append(", testigo1=").append(testigo1);
    sb.append(", testigo2=").append(testigo2);
    sb.append(", autoridad=").append(autoridad);
    sb.append('}');
    return sb.toString();
}
}

```

Controlador

Clase AbstractControlador

```
package ec.edu.upes.controlador;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

/**
 *
 * @author user
 */
public abstract class AbstractControlador<E> {

    private List<E> lista;

    String ruta;

    public AbstractControlador(String ruta) {
        lista = new ArrayList();
        this.ruta = ruta;

        cargarDatos();
    }

    public final void cargarDatos() {

        try {
            FileInputStream archivo = new FileInputStream(ruta);
            ObjectInputStream datos = new ObjectInputStream(archivo);
            lista = (List<E>) datos.readObject();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void guardarDatos(String ruta) throws FileNotFoundException, IOException
    {
        FileOutputStream archivo = new FileOutputStream(ruta);
        ObjectOutputStream datos = new ObjectOutputStream(archivo);
        datos.writeObject(lista);
    }

    public boolean crear(E objeto) {

        if (validar(objeto) == true) {
            return lista.add(objeto);
        }
    }
}
```

```

    }
    return false;

}

public Optional<E> buscar(E comparar) {
    return lista.stream().filter(objeto -> objeto.equals(comparar)).findFirst();
}

public int posicion(E objetoC) {
    for (int i = 0; i < lista.size(); i++) {
        E objetoL = lista.get(i);
        if (objetoL.equals(objetoC)) {
            return i;
        }
    }
    return -1;
}

public boolean eliminar(E objeto) {
    Optional<E> buscar = buscar(objeto);
    E objetoE = buscar.get();
    if (objetoE != null) {
        System.out.println("Verdadero");
        return lista.remove(objetoE);
    }
    System.out.println("Falso");
    return false;
}

public boolean actualizar(E objetoA) {
    int pos = posicion(objetoA);
    if (pos >= 0) {
        lista.set(pos, objetoA);
        System.out.println("TRUE");
        return true;
    }
    System.out.println("FALSE");
    return false;
}

public abstract boolean validar(E objeto);

public abstract int generarId();

public List<E> getLista() {
    return lista;
}

public void setLista(List<E> lista) {
    this.lista = lista;
}
}

```

Clase controladorPersona

```
package ec.edu.ups.controlador;

import ec.edu.ups.modelo.Persona;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

/**
 *
 * @author user
 */
public class controladorPersona extends AbstractControlador<Persona> {

    public controladorPersona(String ruta) {
        super(ruta);
    }

    @Override
    public boolean validar(Persona objeto) {
        int suma = 0;
        String x = objeto.getCedula();
        if (x.length() == 9) {
            return false;
        } else {
            int a[] = new int[x.length() / 2];
            int b[] = new int[(x.length() / 2)];
            int c = 0;
            int d = 1;
            for (int i = 0; i < x.length() / 2; i++) {
                a[i] = Integer.parseInt(String.valueOf(x.charAt(c)));
                c = c + 2;
                if (i < (x.length() / 2) - 1) {
                    b[i] = Integer.parseInt(String.valueOf(x.charAt(d)));
                    d = d + 2;
                }
            }

            for (int i = 0; i < a.length; i++) {
                a[i] = a[i] * 2;
                if (a[i] > 9) {
                    a[i] = a[i] - 9;
                }
                suma = suma + a[i] + b[i];
            }
            int aux = suma / 10;
            int dec = (aux + 1) * 10;
            if ((dec - suma) == Integer.parseInt(String.valueOf(x.charAt(x.length()
- 1)))) {
                return true;
            } else if (suma % 10 == 0 && x.charAt(x.length() - 1) == '0') {
                return true;
            } else {
                return false;
            }
        }
    }
}
```

```

}

@Override
public int generarId() {
    return 0;
}

// RETORNA LISTA DE OBJETOS DE TIPO PERSONA
public List<Persona> personas() {

    List<Persona> listaP = new ArrayList();
    Persona persona;
    Iterator i = super.getList().iterator();
    while (i.hasNext()) {
        persona = (Persona) i.next();
        listaP.add(persona);
    }
    return listaP;
}

```

Clase controladorAutoridad

```

public class controladorAutoridad extends AbstractControlador<Autoridad> {

    private Autoridad autoridad;

    public controladorAutoridad(String ruta) {
        super(ruta);
    }

    @Override
    public boolean validar(Autoridad objeto) {
        if (objeto.getRol().equals("Autoridad")) {
            int suma = 0;
            String x = objeto.getCedula();
            if (x.length() == 9) {
                return false;
            } else {
                int a[] = new int[x.length() / 2];
                int b[] = new int[(x.length() / 2)];
                int c = 0;
                int d = 1;
                for (int i = 0; i < x.length() / 2; i++) {
                    a[i] = Integer.parseInt(String.valueOf(x.charAt(c)));
                    c = c + 2;
                    if (i < (x.length() / 2) - 1) {
                        b[i] = Integer.parseInt(String.valueOf(x.charAt(d)));
                        d = d + 2;
                    }
                }

                for (int i = 0; i < a.length; i++) {
                    a[i] = a[i] * 2;
                    if (a[i] > 9) {
                        a[i] = a[i] - 9;
                    }
                    suma = suma + a[i] + b[i];
                }
            }
        }
    }
}

```

```

        int aux = suma / 10;
        int dec = (aux + 1) * 10;
        if ((dec - suma) == Integer.parseInt(String.valueOf(x.cha-
rAt(x.length() - 1)))) {
            return true;
        } else if (suma % 10 == 0 && x.charAt(x.length() - 1) == '0') {
            return true;
        } else {
            return false;
        }
    }
} else {
    return false;
}

}

@Override
public int generarId() {
    return 0;
}

public boolean iniciarSesion(String correo, String pass) {

    for (Autoridad usu : super.getLista()) {
        Autoridad u = (Autoridad) usu;
        if (u.getCorreo().equals(correo) && u.getPass().equals(pass)) {
            this.autoridad = u;
            return true;
        }
    }
    return false;
}

public Autoridad getAutoridad() {
    return autoridad;
}

public void setAutoridad(Autoridad autoridad) {
    this.autoridad = autoridad;
}

}

```

ControladorMatrimonio

```

package ec.edu.ups.controlador;

import ec.edu.ups.modelo.Matrimonio;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

/**
 *
 * @author user
 */
public class controladorMatrimonio extends AbstractControlador<Matrimonio> {

```

```

public controladorMatrimonio(String ruta) {
    super(ruta);
}

@Override
public boolean validar(Matrimonio objeto) {
    if (objeto.getContrayente().getRol().equals("Contrayente") && objeto.get-
    Contreyente2().getRol().equals("Contrayente")) {
        if (objeto.getContrayente().getEstadoCivil() != "Casado" && objeto.get-
    Contreyente2().getEstadoCivil() != "Casado") {
            return true;
        }
    }

    return false;
}

@Override
public int generarId() {
    List<Matrimonio> temp = new ArrayList();
    for (Matrimonio matrimonio : super.getList()) {
        Matrimonio m = (Matrimonio) matrimonio;
        temp.add(m);
    }

    if (temp.size() > 0 && temp != null) {
        return temp.get(temp.size() - 1).getCodigoM() + 1;
    } else {
        return 1;
    }
}

// RETORNA LISTA DE OBJETOS DE TIPO Matrimonio
public List<Matrimonio> registros() {

    List<Matrimonio> listaM = new ArrayList();
    Matrimonio matrimonio;
    Iterator i = super.getList().iterator();
    while (i.hasNext()) {
        matrimonio = (Matrimonio) i.next();
        listaM.add(matrimonio);
    }
    return listaM;
}

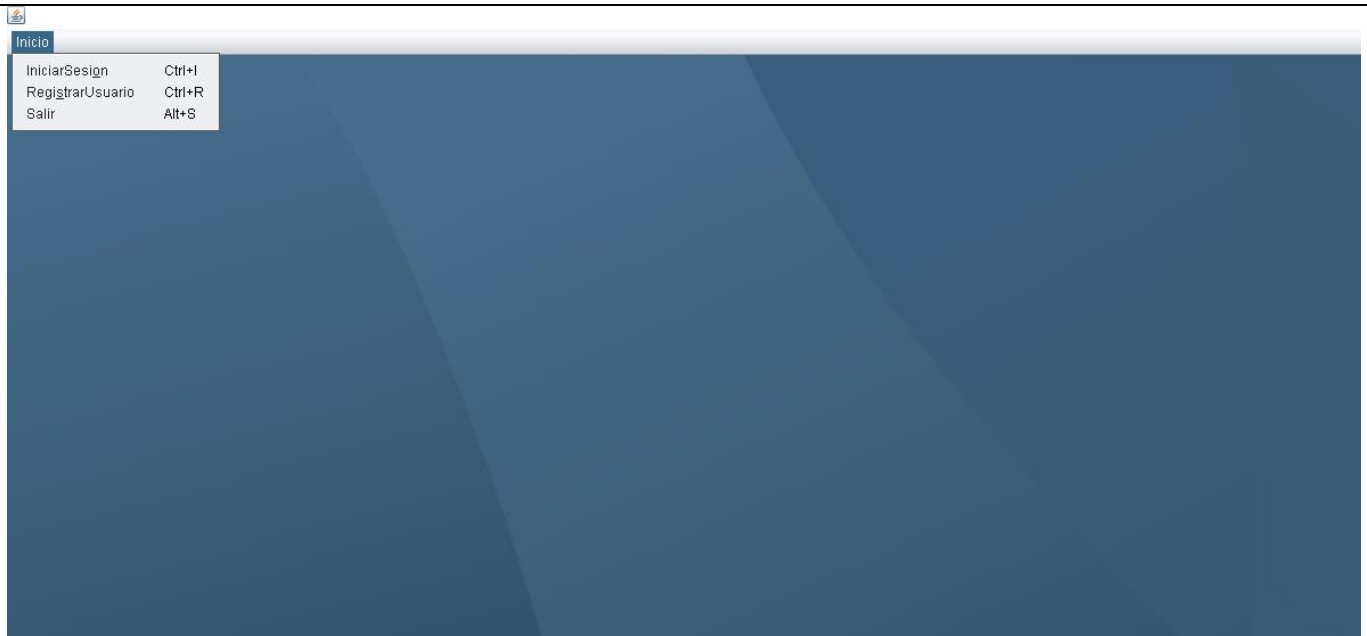
}

```

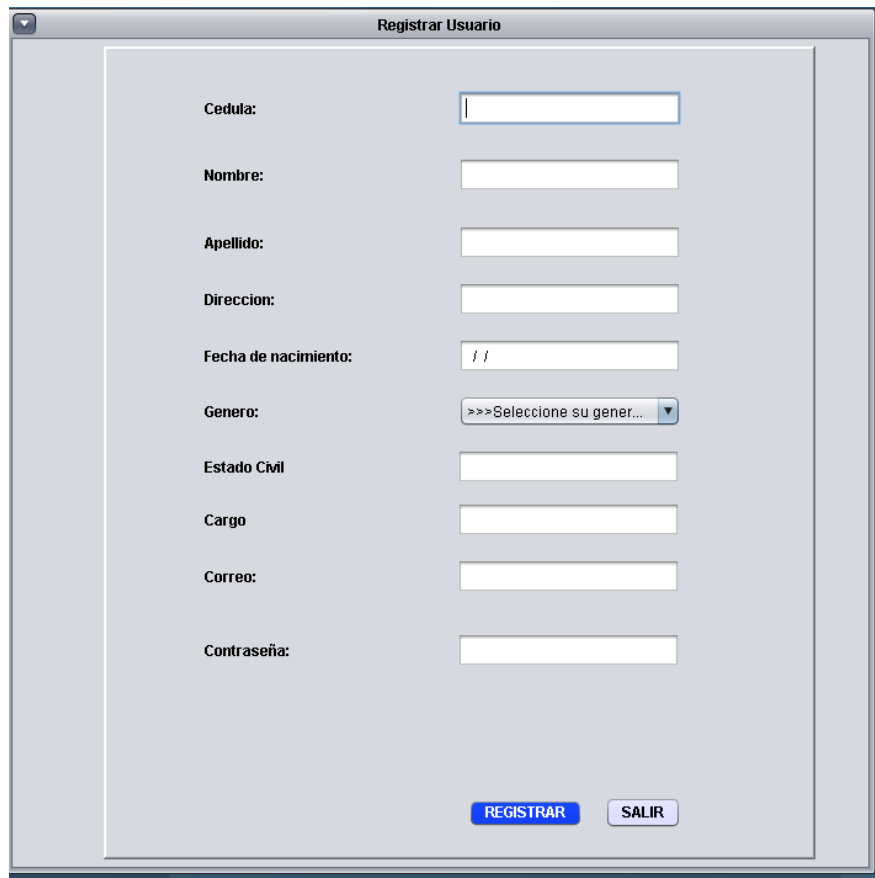
Vista

Diseños de Ventanas

VentanaPrincipal



Ventana Registrar Usuario (Autoridad)

A screenshot of a 'Registrar Usuario' window. The window has a title bar with a dropdown arrow and the text 'Registrar Usuario'. Inside, there is a form with the following fields and labels: 'Cedula:' followed by a text input field; 'Nombre:' followed by a text input field; 'Apellido:' followed by a text input field; 'Direccion:' followed by a text input field; 'Fecha de nacimiento:' followed by a date input field showing ' / /'; 'Genero:' followed by a dropdown menu with the text '>>>Seleccione su gener...'; 'Estado Civil' followed by a text input field; 'Cargo' followed by a text input field; 'Correo:' followed by a text input field; and 'Contraseña:' followed by a text input field. At the bottom right of the form, there are two buttons: 'REGISTRAR' (highlighted in blue) and 'SALIR'.

Ventana de inicio de sesión de los usuarios (autoridades)

INICIAR SESION

Correo:

Contraseña:

Ventana para generar ficha de matrimonio

CODIGO DE REGISTRO: 0 **REGISTRO DE MATRIMONIO** AUTORIDAD: Juan Cordova JUEZ PROVI

FECHA: Sat... LUGAR:

Cedula contrayente: <input type="text"/>	<input type="button" value="BUSCAR"/>
Nombre: <input type="text"/>	
Apellido: <input type="text"/>	
Direccion: <input type="text"/>	
Estado Civil: <input type="text"/>	
Fecha de nacimiento: <input type="text"/>	
Genero: <input type="text"/>	
Cedula Testigo: <input type="text"/>	<input type="button" value="BUSCAR"/>
Nombres del testigo :	

Cedula contrayente: <input type="text"/>	<input type="button" value="BUSCAR"/>
Nombre: <input type="text"/>	
Apellido: <input type="text"/>	
Direccion: <input type="text"/>	
Estado Civil: <input type="text"/>	
Fecha de nacimiento: <input type="text"/>	
Genero: <input type="text"/>	
Cedula Testigo: <input type="text"/>	<input type="button" value="BUSCAR"/>
Nombres del testigo :	

Ventana para la gestión de Persona

Gestion Persona

Cedula:

Nombre:

Apellido:

Direccion:

Fecha de nacimiento:

Genero:

Estado Civil:

AGREGAR **MODIFICAR** **ELIMINAR** **LISTAR** **SALIR**

Cedula	Nombre	Apellido	Direccion	Fecha de Nacimie...	Genero	Estado Civil

Ventana listar Registros por usuario logeado

Listado de registros de mstrimonios por Usuario Autoridad logeado

Codigo	Lugar	Fecha	Contrayente 1	Contrayente 2	Testigo 1	Testigo 2	Autoridad

RESULTADO(S) OBTENIDO(S):

Se interpreto de forma correcta los algoritmos de programación y su aplicabilidad, se identificó correctamente las herramientas de programación a aplicar

CONCLUSIONES:

Se identifico estructuras para una programación más sencilla y acoplada mediante la ayuda de programación genérica, Reflexión en Java y patrones de diseño.

RECOMENDACIONES:

Investigar aplicaciones que engloben registros y programación genérica para poder tener una base estructural para las siguientes.

Nombre de estudiante: JUAN JOSE CORDOVA CALLE

Firma de estudiante:

