
	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 tecnicas 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. Ademas de gestionar la seguridad a traves de un sistema de Usuarios y Autentificación.

Calificación:

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

2. Informe de Actividades:

- Planteamiento y descripción del problema.
- Diagramas de Clases.

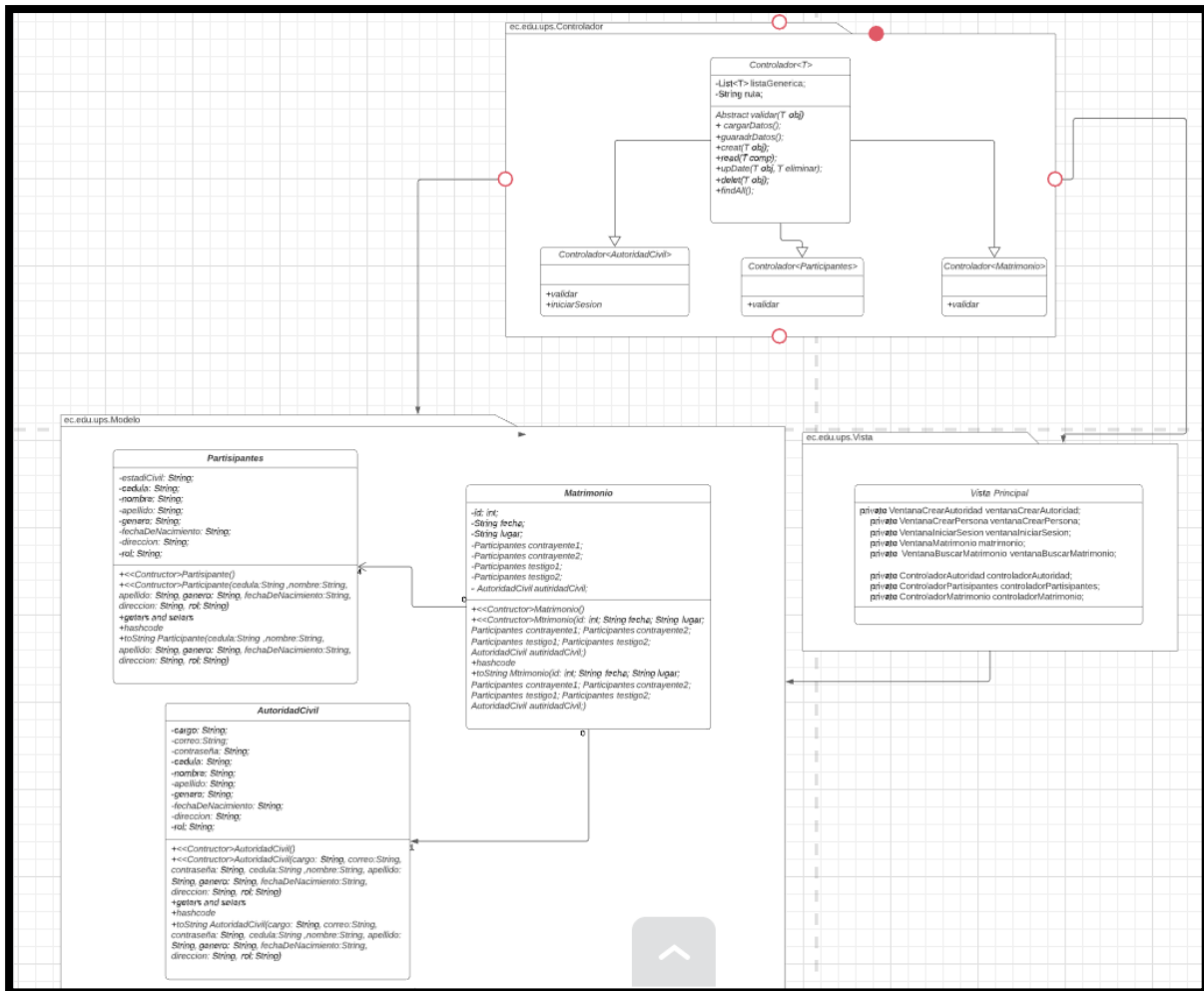
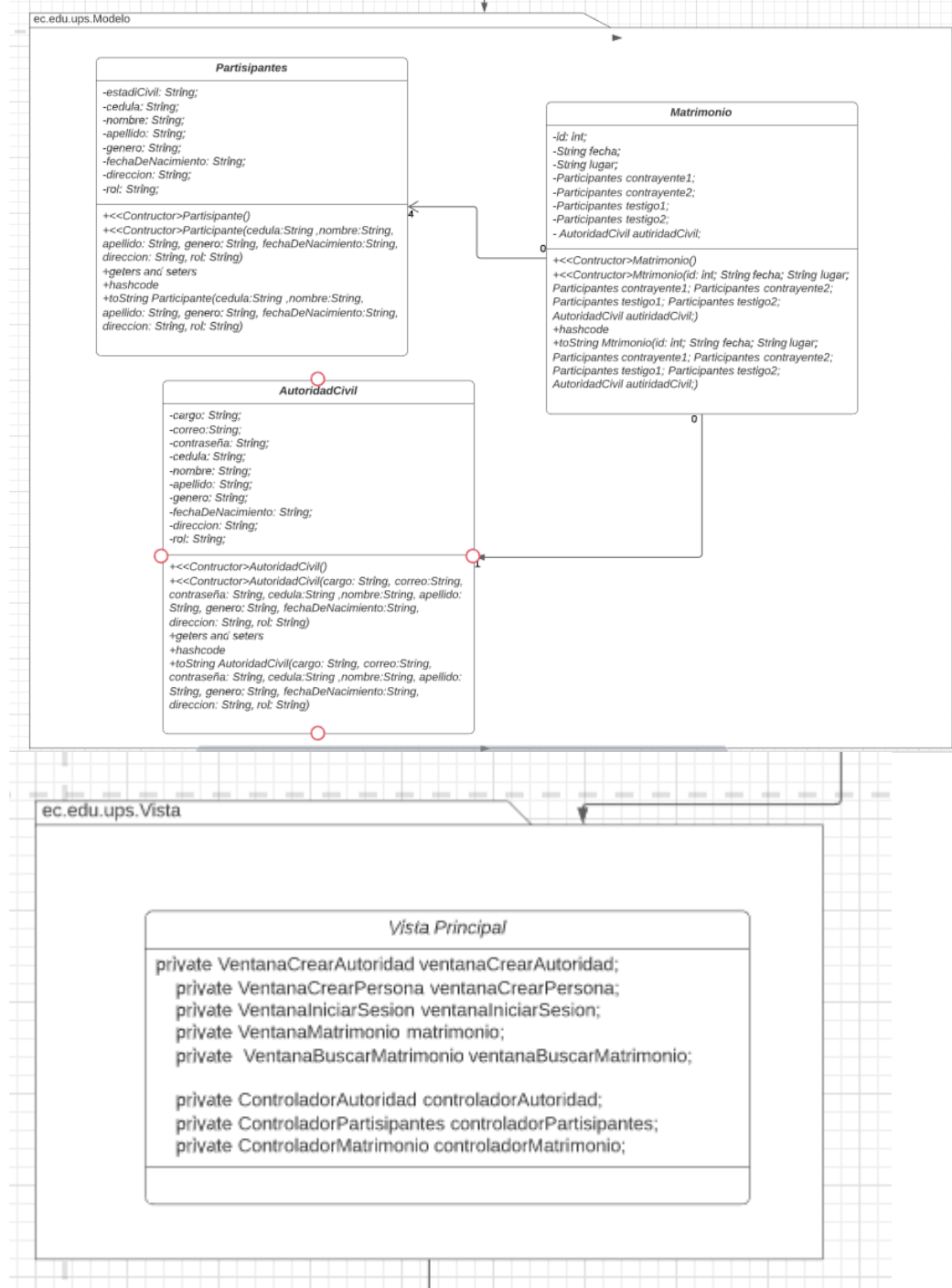
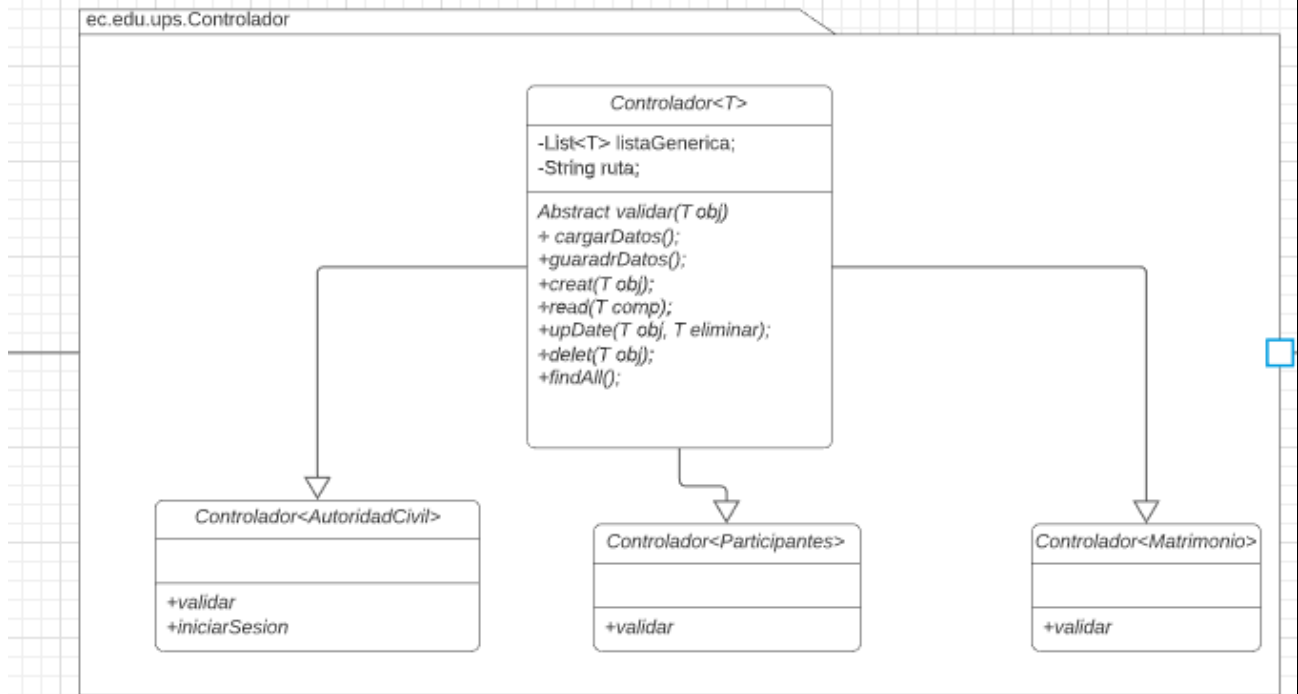


Diagrama uml por paquetes



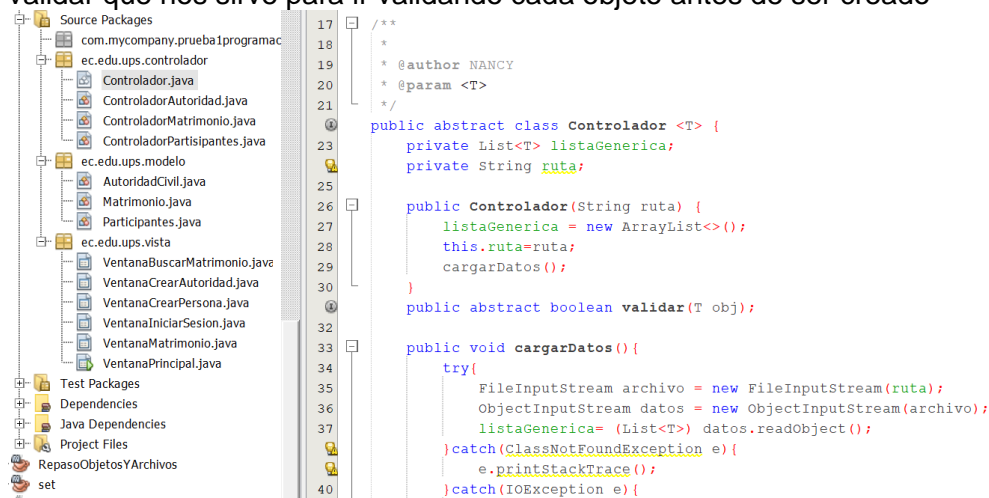


Enlace del diagrama uml realizado en lucidchar https://lucid.app/lucidchart/796ba086-6780-47cb-b8ff-6e66731383b0/view?page=0_0?folder_id=home&brower=icon

- Patrón de diseño aplicado

Factory Method

En este caso usamos el patrón de diseño Factory Method debido a que el controlador es una clase generadora y este patrón de diseño sirve para instanciar un método que después será utilizado por las demás clases es decir un método abstracto como podemos ver en este caso el método validar que nos sirve para ir validando cada objeto antes de ser creado



También podemos observar en la imagen el uso del patrón de diseño MVC

- Descripción de la solución y pasos seguidos.

El código consta con tres paquetes el primero que es el modelo el cual contiene las clases de Autoridades Civiles <https://github.com/RomelAvila2001/Prueba-unidad-1-Programacion-Aplicada/blob/master/src/main/java/ec/edu/ups/modelo/AutoridadCivil.java> en el link se puede ver el código pero esta clase es una de las partes importantes del matrimonio ya que son las personas que tienen que iniciar sesión para poder oficiar un matrimonio

En el mismo modelo tenemos la clase participantes <https://github.com/RomelAvila2001/Prueba-unidad-1-Programacion-Aplicada/blob/master/src/main/java/ec/edu/ups/modelo/Participantes.java> en el link se encuentra el código en esta clase tenemos así mismo varios atributos que hacen u a una persona con un detalle que es el estado civil y un rol que tienen que cumplir en el matrimonio entonces aquí especificamos si van a ser testigos o van a ser contrayentes

Y por último en el modelo tenemos la clase matrimonio <https://github.com/RomelAvila2001/Prueba-unidad-1-Programacion-Aplicada/blob/master/src/main/java/ec/edu/ups/modelo/Matrimonio.java> en el link encontramos el código y en esta clase es donde armamos lo que es el matrimonio como tal ya que aquí tenemos como atributos a dos participantes que hacen de contrayentes dos participantes que hacen de testigos lugar y fecha y por último la autoridad ya que sin estos atributos no podríamos generar un matrimonio

Después de esto tenemos el paquete controlador <https://github.com/RomelAvila2001/Prueba-unidad-1-Programacion-Aplicada/tree/master/src/main/java/ec/edu/ups/controlador> el cual como ya mencionamos antes contiene el patrón de diseño factory method que nos ayuda con el método abstracto de validar y así mismo en esta clase tenemos los métodos del cruz que al ser una clase genérica nos permite funcionar con varias clases a la vez sin necesidad de crear más métodos crud en otros controladores

```
public abstract class Controlador <T> {
    private List<T> listaGenerica;
    private String ruta;

    public Controlador(String ruta) {
        listaGenerica = new ArrayList<>();
        this.ruta=ruta;
        cargarDatos();
    }
}
```

Podemos observar como el controlador si es una clase genérica

Así mismo tenemos los demás controladores los cuales pasan como a la clase genérica los distintos tipos de clase con los cuales trabajar

Como vamos a ver en el código ahora

```
public class ControladorAutoridad extends Controlador<AutoridadCivil> {
```

```
    public ControladorAutoridad(String ruta) {
        super(ruta);
    }
}
```

@Override

```
public boolean validar(AutoridadCivil obj) {  
    return true;  
}  
  
public AutoridadCivil iniciarSesion(String correo, String contrase) {  
    for (var autoridad : (List<AutoridadCivil>) findAll()) {  
        if (autoridad.getCorreo().equals(correo) && autoridad.getContraseña().equals(contrase)) {  
            return autoridad;  
        }  
    }  
    return null;  
}  
  
}  
  
public class ControladorMatrimonio extends Controlador<Matrimonio>{  
  
    public ControladorMatrimonio(String ruta) {  
        super(ruta);  
    }  
  
    @Override  
    public boolean validar(Matrimonio obj) {  
  
        if(obj.getContrayente1().getEstadoCivil().equalsIgnoreCase("Casado")  
obj.getContrayente2().getEstadoCivil().equalsIgnoreCase("Casado") ){  
            return false;  
        }  
        return true;  
    }  
}
```

||

```

public int cargarCodigo(){
    if (findAll().size() > 0) {
        return findAll().size() + 1;
    } else {
        return 1;
    }
}
}

public class ControladorParticipantes extends Controlador<Participantes> {

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

    @Override
    public boolean validar(Participantes obj) {
        return true;
    }

}

```

Uso de archivos objetos para guardar los datos

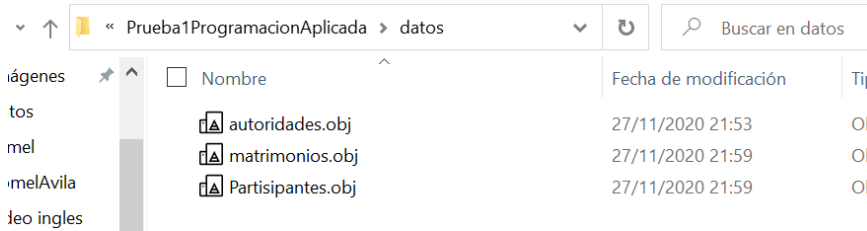
```

public void cargarDatos() {
    try {
        FileInputStream archivo = new FileInputStream(ruta);
        ObjectInputStream datos = new ObjectInputStream(archivo);
        listaGenerica = (List<T>) 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(listaGenerica);
}

```

Son los métodos para guardar la información y obtener la información de una archivo objeto



Nombre	Fecha de modificación	Tip
autoridades.obj	27/11/2020 21:53	Obj
matrimonios.obj	27/11/2020 21:59	Obj
Participantes.obj	27/11/2020 21:59	Obj

Podemos ver a los archivos objetos creados en una carpeta dentro del proyecto

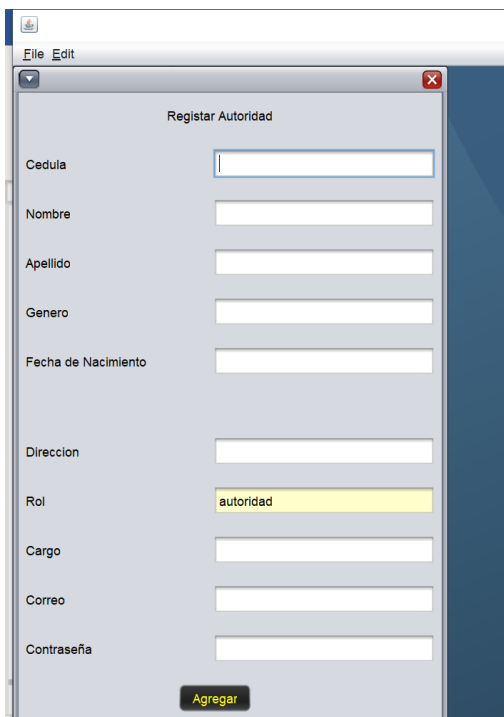
Entonces estos son los códigos de las clases que se encuentran en el paquete controlador

Finalmente tenemos lo que son las vistas donde en la vista principal instanciamos los controladores para ser enviados a las otras vistas

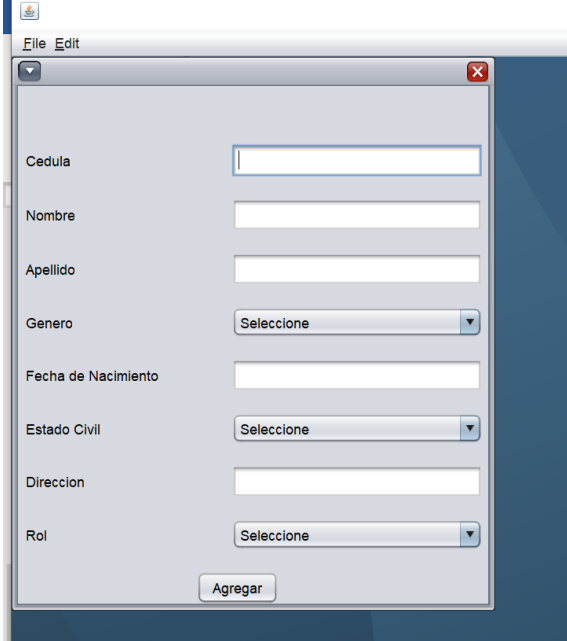
Interface grafica



Al ejecutar el programa se nos presenta la siguiente pantalla donde podemos escoger en el menú las opciones para ir creando los elementos del matrimonio



Al presionar registrar autoridad se nos abre la siguiente ventana que nos ayudara a crear una autoridad validando que los campos esten llenos caso contrario no lo hará

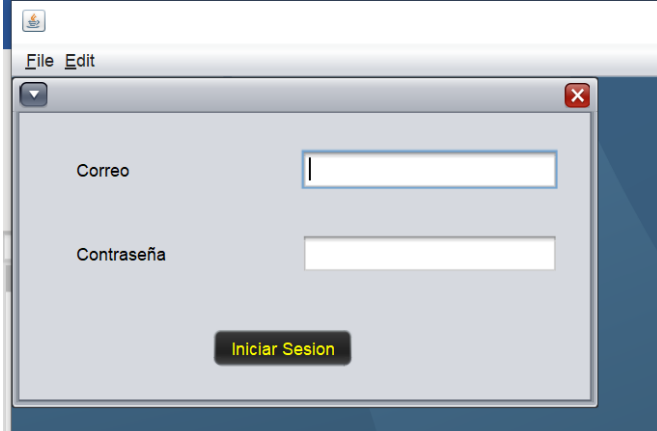


Formulario para registrar una autoridad. El formulario contiene los siguientes campos:

- Cedula: Campo de texto.
- Nombre: Campo de texto.
- Apellido: Campo de texto.
- Genero: Selector de lista desplegable con la opción "Selecciona".
- Fecha de Nacimiento: Campo de texto.
- Estado Civil: Selector de lista desplegable con la opción "Selecciona".
- Direccion: Campo de texto.
- Rol: Selector de lista desplegable con la opción "Selecciona".

En la parte inferior del formulario hay un botón "Agregar".

Al presionar registrar persona se nos abre la venta para poder crear una persona y ser guardada en el archivo obj

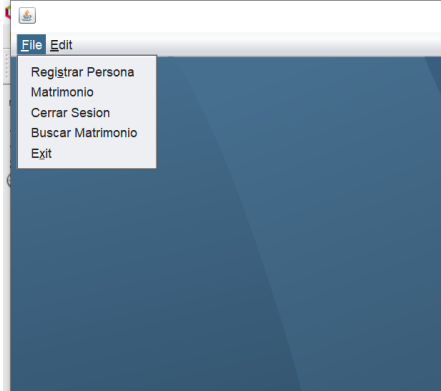


Formulario para iniciar sesión. El formulario contiene los siguientes campos:

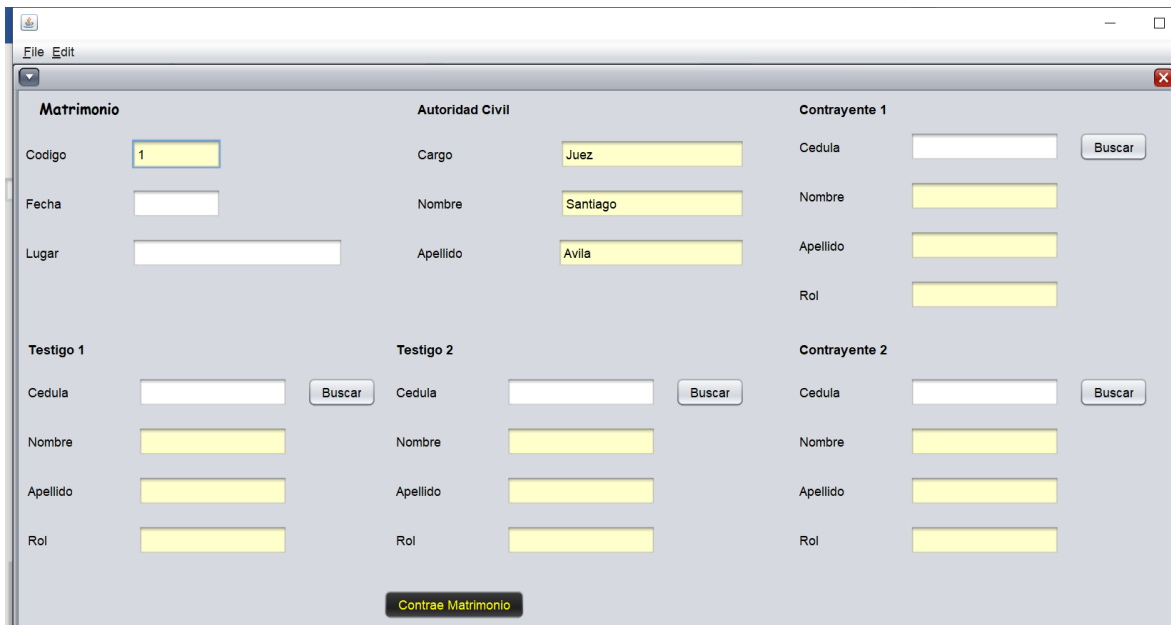
- Correo: Campo de texto.
- Contraseña: Campo de texto.

En la parte inferior del formulario hay un botón "Iniciar Sesión".

La ventana para iniciar sesión con la cual solo las autoridades pueden hacerlo ya que son las únicas que tienen correo y contraseña y una vez accedido se despliega otro menu itm



Vemos el nuevo menú itm que nos ayuda a crear matrimonios y cerrar sesión



Finalmente vemos la ventana para poder crear los matrimonios en donde vamos buscando cedula acidula a cedula cada participante y se valida el campo de rol para ver si es el correcto

RESULTADO(S) OBTENIDO(S):

- logramos realizar un programa aplicando varios patrones de diseño a la vez
- Conseguimos guardar en archivos obj los datos del programa
- Utilizamos programación genérica para optimizar el código al igual que el uso de Streams

CONCLUSIONES:

- En conclusión este trabajo a sido de mucha ayuda para profundizar mas los conceptos aprendidos en la unidad numero 1

RECOMENDACIONES:

Dar un poquito mas de tiempo para la solución de la evaluación

Estudiantes: Romel Ávila

Firma:

