
	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 Activades:


- Planteamiento y descipcion del problema.
- Digramas de Clases.
- Patron de diseño aplicado
- Descripcion de la solucion y pasos seguidos.
- Conclusiones y recomendaciones.
- Resultados.

RESULTADO(S) OBTENIDO(S):

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

CONCLUSIONES:

- Los estudiantes identifican las principales estructuras para la creacion de sistemas informaticos.
- Los estudiantes implementan soluciones graficas en sistemas.
- Los estudiantes estan 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: _____

CARRERA: Ingeniería en Computación

ASIGNATURA: Programación Aplicada

NRO. PRÁCTICA:

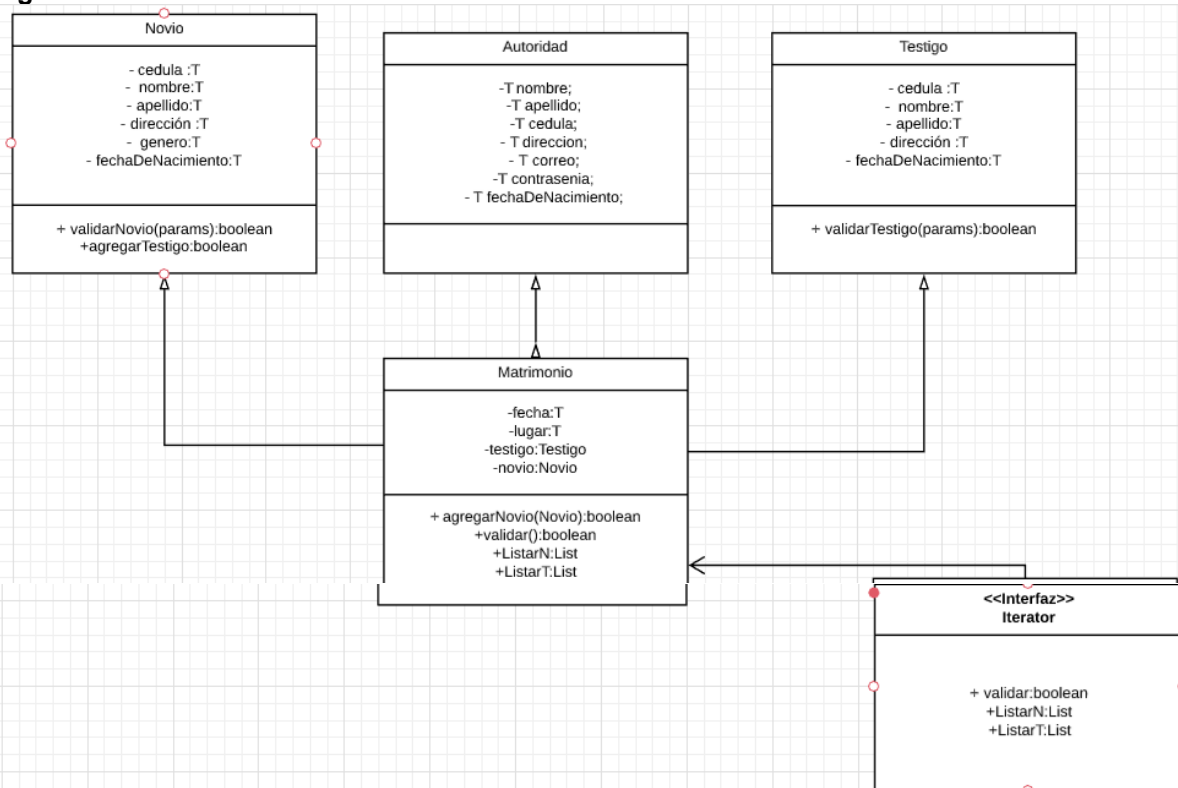
TÍTULO PRÁCTICA: Prueba práctica

OBJETIVO ALCANZADO:

Reforzar los conocimientos adquiridos en clase sobre la programación aplicada (Java 8, Programación Genérica, Reflexión y Patrones de Diseño) en un contexto real.

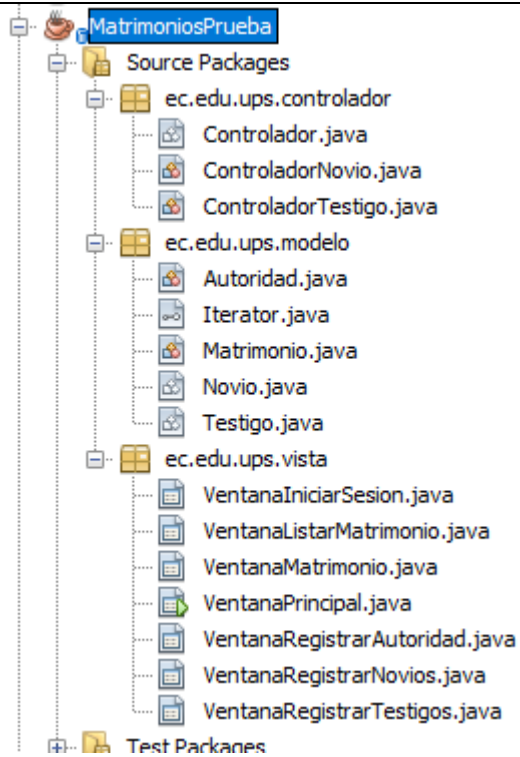
ACTIVIDADES DESARROLLADAS

1. Diagrama de clases:



2.Programa:

Creamos el Proyecto con sus respectivos paquetes:



Paquete modelo:
Clase "Novio":

```
public abstract class Novio <T>{

    private T nombre;
    private T apellido;
    private T cedula;
    private T direccion;
    private T genero;
    private T fechaDeNacimiento;
    private Testigo testigo;
    private Autoridad autoridad;
    //Agregacion
    private List<Testigo> testigos;

    public Novio() {
        testigos = new ArrayList<>();
    }

    public Novio(T nombre, T apellido, T cedula, T direccion, T genero, T fechaDeNacimiento) {
        this.nombre = nombre;
        this.apellido = apellido;
        this.cedula = cedula;
        this.direccion = direccion;
        this.genero = genero;
        this.fechaDeNacimiento = fechaDeNacimiento;
        testigos = new ArrayList<>();
    }
}
```

```
public T getNombre() {
    return nombre;
}

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

public T getApellido() {
    return apellido;
}

public void setApellido(T apellido) {
    this.apellido = apellido;
}

public T getCedula() {
    return cedula;
}

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

public T getDireccion() {
    return direccion;
}

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

public T getGenero() {
    return genero;
}

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

public T getFechaDeNacimiento() {
    return fechaDeNacimiento;
}

public void setFechaDeNacimiento(T fechaDeNacimiento) {
    this.fechaDeNacimiento = fechaDeNacimiento;
}

public boolean agregarTestigo(Testigo t) {
    return this.testigos.add(new Testigo(t.getNombre(), t.getApellido(), t.getCedula(), t.getDireccion(), t.getFechaDeNa
});

public boolean validarM(){
    return true;
}
```

Clase "Testigo":


```
public abstract class Testigo <T>{

    private T nombre;
    private T apellido;
    private T cedula;
    private T direccion;
    private T fechaDeNacimiento;

    public Testigo() {
    }

    public Testigo(T nombre, T apellido, T cedula, T direccion, T fechaDeNacimiento) {
        this.nombre = nombre;
        this.apellido = apellido;
        this.cedula = cedula;
        this.direccion = direccion;
        this.fechaDeNacimiento = fechaDeNacimiento;
    }

    public T getNombre() {
        return nombre;
    }

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

    public T getApellido() {
        return apellido;
    }

    public void setApellido(T apellido) {
        this.apellido = apellido;
    }

    public T getCedula() {
        return cedula;
    }

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

    public T getDireccion() {
        return direccion;
    }

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

```
public T getFechaDeNacimiento() {  
    return fechaDeNacimiento;  
}  
  
public void setFechaDeNacimiento(T fechaDeNacimiento) {  
    this.fechaDeNacimiento = fechaDeNacimiento;  
}  
  
@Override  
public String toString() {  
    return "Testigo{" + "nombre=" + nombre + ", apellido=" + apellido + ", cedula=" + cedula + ", direccion=" + direccion + "  
}  
}
```

Clase "Autoridad":

```
public class Autoridad <T> {  
  
    private T nombre;  
    private T apellido;  
    private T cedula;  
    private T direccion;  
    private T correo;  
    private T contrasenia;  
    private T fechaDeNacimiento;  
  
    public Autoridad() {  
    }  
  
    public Autoridad(T nombre, T apellido, T cedula, T direccion, T correo, T contrasenia, T fechaDeNacimiento) {  
        this.nombre = nombre;  
        this.apellido = apellido;  
        this.cedula = cedula;  
        this.direccion = direccion;  
        this.correo = correo;  
        this.contrasenia = contrasenia;  
        this.fechaDeNacimiento = fechaDeNacimiento;  
    }  
  
    public T getNombre() {  
        return nombre;  
    }  
  
    public void setNombre(T nombre) {  
        this.nombre = nombre;  
    }  
  
    public T getApellido() {  
        return apellido;  
    }  
  
    public void setApellido(T apellido) {  
        this.apellido = apellido;  
    }  
  
    public T getCedula() {  
        return cedula;  
    }  
  
    public void setCedula(T cedula) {  
        this.cedula = cedula;  
    }  
  
    public T getDireccion() {  
        return direccion;  
    }  
  
    public void setDireccion(T direccion) {  
        this.direccion = direccion;  
    }  
}
```

```

public T getFechaDeNacimiento() {
    return fechaDeNacimiento;
}

public void setFechaDeNacimiento(T fechaDeNacimiento) {
    this.fechaDeNacimiento = fechaDeNacimiento;
}

public T getCorreo() {
    return correo;
}

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

public T getContrasenia() {
    return contrasenia;
}

public void setContrasenia(T contrasenia) {
    this.contrasenia = contrasenia;
}

@Override
public String toString() {
    return "Autoridad{" + "nombre=" + nombre + ", apellido=" + apellido + ", cedula=" + cedula + ", direccion=" + d
}

```

Clase "Matrimonio":

```
public class Matrimonio <T> implements Iterator{

    private T fecha;
    private T lugar;
    private Novio novio;
    private Testigo testigo;
    private Autoridad autoridad;

    //Agregacion
    private List<Novio> novios;
    private List<Testigo> testigos;

    public Matrimonio() {
        novios = new ArrayList<>();
        testigos = new ArrayList<>();
    }

    public Matrimonio( T fecha, T lugar) {

        this.fecha = fecha;
        this.lugar = lugar;
        novios = new ArrayList<>();
        testigos = new ArrayList<>();
    }

    public T getFecha() {
        return fecha;
    }

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

    public T getLugar() {
        return lugar;
    }

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

    //metodos

    //métodos de la agregación
    public boolean agregarNovio(Novio n) {
        return this.novios.add(new Novio(n.getNombre(),n.getApellido(),n.getCedula(),n.getGenero(),n.getDireccion(),n.get
    }
}
```

```
@Override
public String toString() {
    return fecha + ", lugar=" + lugar + ' ';
}

@Override
public boolean validar() {
    boolean bandera = false;
    for(int i=0;i<novios.size();i++){

        bandera= true;
    }

    for(int i=0;i<testigos.size();i++){

        bandera= true;
    }

    return bandera;
}
```

```
@Override
public List<Novio> listarN() {

    for(int i=0;i<novios.size();i++){
        novios.stream().forEach((n)-> {
            System.out.println(n.getCedula());
            System.out.println(n.getNombre());
            System.out.println(n.getApellido());
            System.out.println(n.getGenero());
            System.out.println(n.getDireccion());
            System.out.println(n.getFechaDeNacimiento());

        });
    }
    return novios;
}

@Override
public List<Testigo> listarT() {

    for(int i=0;i<testigos.size();i++){
        testigos.stream().forEach((t)-> {
            System.out.println(t.getCedula());
            System.out.println(t.getNombre());
            System.out.println(t.getApellido());
            System.out.println(t.getDireccion());
            System.out.println(t.getFechaDeNacimiento());

        });
    }

    return testigos;
}
```

Interfaz "Iterator":

```
*/
public interface Iterator {

    public boolean validar();
    public List<Novio> listarN();
    public List<Testigo> listarT();
}
```

Paquete controlador:

Clase abstracta "Controlador":

```
public abstract class Controlador<T> {

    private List <T> LGenerica;
    private Novio n;

    public List<T> getLGenerica() {
        return LGenerica;
    }

    public void setLGenerica(List<T> LGenerica) {
        this.LGenerica = LGenerica;
    }

    public Controlador(List<T> LGenerica) {
        LGenerica = new ArrayList<>();
    }

    public void registrar(T obj) {

        LGenerica.add(obj);
    }

    public T verPersona(T obj) {

        return (T) LGenerica.stream().filter((e)->e.equals(obj)).findFirst().get();
    }
}
```



```
public void update(T obj) {
    for (int i = 0; i < LGenerica.size(); i++) {
        obj = LGenerica.get(i);
        if (n.getCedula().equals(i)) {
            LGenerica.set(i, obj);
            break;
        }
    }
}

public void delete(T novio) {

    LGenerica.remove(novio);

}

public List<T> findAll() {
    return LGenerica;
}

public void Almacenamiento(String r) throws IOException {
    FileInputStream file = new FileInputStream(r);

    ObjectInputStream datos = new ObjectInputStream(file);

    try {
        LGenerica = (List<T>) datos.readObject();
    } catch (ClassNotFoundException ex) {
        Logger.getLogger(Controlador.class.getName()).log(Level.SEVERE, null, ex);
    }

}

public void guardar(String r) throws IOException {
    FileOutputStream archivo = new FileOutputStream(r);

    ObjectOutputStream datos = new ObjectOutputStream(archivo);
    datos.writeObject(LGenerica);
}

public abstract boolean validar(T obj);
```

Clase "controlador novio":

```
public class ControladorNovio extends Controlador<Novio>{  
  
    public ControladorNovio(List<Novio> list) {  
        super(list);  
    }  
  
    @Override  
    public boolean validar(Novio obj) {  
        return false;  
    }  
}
```

Clase "Controlador Testigo":

```
public class ControladorTestigo extends Controlador<Testigo> {  
  
    public ControladorTestigo(List<Testigo> LGenerica) {  
        super(LGenerica);  
    }  
  
    @Override  
    public boolean validar(Testigo obj) {  
  
        return true;  
    }  
}
```

El paquete vista:

Ventana principal:

Registro Gestión Matrimonio

VentanaRegistrarNovios:

REGISTRO

Persona 1:

Cédula:

Nombre:

Apellido:

Dirección:

Fecha de Nacimiento:

Género:

Persona 2:

Cédula:

Nombre:

Apellido:

Dirección:

Fecha de Nacimiento:

Género:

AGREGAR

VentanaRegistrarTestigos:

REGISTRO			
Testigo 1:		Testigo 2:	
Cédula:	<input type="text"/>	Cédula:	<input type="text"/>
Nombre:	<input type="text"/>	Nombre:	<input type="text"/>
Apellido:	<input type="text"/>	Apellido:	<input type="text"/>
Fecha de Nacimiento:	<input type="text"/>	Fecha de Nacimiento:	<input type="text"/>
Dirección:	<input type="text"/>	Dirección:	<input type="text"/>
<input type="button" value="AGREGAR"/>			



REGISTRO

Autoridad Civil

Cédula:

Nombre:

Apellido:

Dirección:


Fecha de Nacimiento:

Correo Electrónico:

Contraseña:

AGREGAR

VentanaMatrimonio:



REGISTRO MATRIMONIO

Fecha:

Lugar:

REGISTRAR

VentanaIniciarSesion:

REGISTRO

Autoridad Civil

Cédula:

Nombre:

Apellido:

Dirección:

Fecha de Nacimiento:

Correo Electrónico:

Contraseña:

AGREGAR

Ventana Listar Matrimonio:

GESTIÓN MATRIMONIO

Cedula	Nombre	Apellido	Género	Dirección	F.Nacimiento

Cedula	Nombre	Apellido	Dirección	F.Nacimiento

LISTAR

RESULTADO(S) OBTENIDO(S):

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

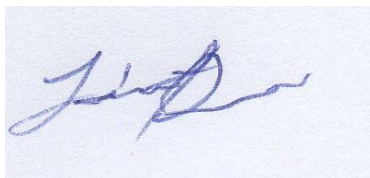
CONCLUSIONES:

- Los estudiantes identifican las principales estructuras para la creacion de sistemas informaticos.
- Los estudiantes implementan soluciones graficas en sistemas.
- Los estudiantes estan en la capacidad de implementar la persistencia en archivos.

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.**

Nombre de estudiante: _____ Lisseth Reinoso _____



Firma de estudiante: