
	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:

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

CARRERA:

ASIGNATURA:

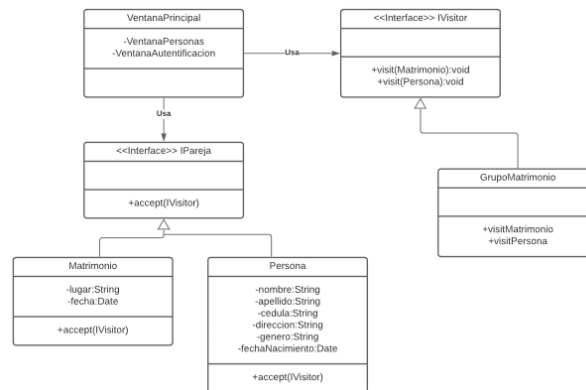
NRO. PRÁCTICA:

TÍTULO PRÁCTICA:

OBJETIVO ALCANZADO:

ACTIVIDADES DESARROLLADAS

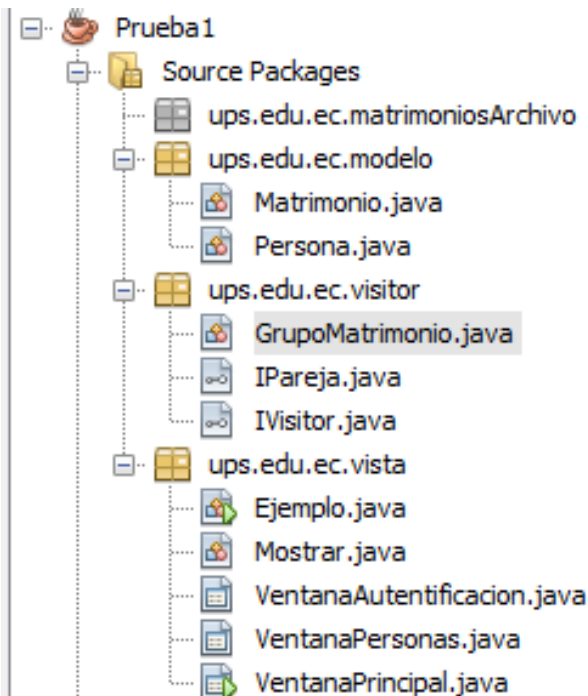
1.



Según el libro de GoF este patrón permite añadir funcionalidades a una clase sin tener que modificarla, siendo usado para manejar algoritmos, relaciones y responsabilidades entre objetos.

Así pues, nos resultará útil cuando necesitemos realizar operaciones distintas y no relacionadas sobre una estructura de objetos, aunque si lo utilizamos y luego tenemos que modificar alguna de las clases implicadas, hemos de tener en cuenta que se produce cierto nivel de acoplamiento entre ellas.

2.



Primero crearemos los paquetes con sus respectivas clases que nos ayudaran en el funcionamiento del programa

3.

```

L  ... */
    public interface IPareja {

        public void accept(IVisitor visitor);

    }

-  ... */
    public interface IVisitor {

        public void visit(Matrimonio matrimonio);

        public void visit(Persona persona);

    }

```

Creamos las interfaces necesarias para implementar el patrón de diseño visitor

```

    ArrayList<Persona> pareja;
    ArrayList<Persona> testigos;
    Persona autoridad;
    Date fecha;
    String lugar;

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

    public Matrimonio(ArrayList<Persona> pareja,
        this.pareja = pareja;
        this.testigos = testigos;
        this.autoridad = autoridad;
        this.fecha = fecha;
        this.lugar = lugar;
    }

    public ArrayList<Persona> getPareja() {
        return pareja;
    }

```

```

@Override
public void accept(IVisitor visitor) {

    visitor.visit(this);

    pareja.stream().forEach(n -> n.accept(visitor));
}

```

```

public class Persona implements IPareja{

    private String nombre;
    private String apellido;
    private String cedula;
    private String direccion;
    private String genero;
    private Date fechaNacimiento;

    public Persona() {
    }

    public Persona(String nombre) {
        this.nombre = nombre;
    }

    public Persona(String nombre, String apellido,
        String cedula, String direccion, String genero,
        Date fechaNacimiento) {
        this.nombre = nombre;
        this.apellido = apellido;
        this.cedula = cedula;
        this.direccion = direccion;
        this.genero = genero;
        this.fechaNacimiento = fechaNacimiento;
    }
}

```

Creamos los modelos del matrimonio y persona con sus constructores, getters y setters. Al igual que estos implementan de IPareja por lo cual tienen sus métodos abstractos

4.

```

public class GrupoMatrimonio<T> implements IPareja{

    List<T> matrimonios;

    public GrupoMatrimonio() {
        matrimonios = new ArrayList<>();
    }

    public List<T> getMatrimonios() {
        return matrimonios;
    }

    public void setMatrimonios(List<T> matrimonios) {
        this.matrimonios = matrimonios;
    }

    public void agregarMatrimonio(T t){
        this.matrimonios.add(t);
    }
}

```

Aplicando programación generica podremos agregar los matrimonios a la clase grupomatrimonios

5.

```
    */  
    GrupoMatrimonio g;  
  
    private VentanaAutenticacion ventanaAutenticacion;  
    private VentanaPersonas ventanaMatrimonios;  
  
    public VentanaPrincipal() {  
        initComponents();  
  
        g = new GrupoMatrimonio();  
  
        ventanaAutenticacion = new VentanaAutenticacion();  
        ventanaMatrimonios = new VentanaPersonas(g);  
    }  
  
private void jMenuItem1ActionPerformed(java.awt.event.ActionEvent evt) {  
    try {  
        g = ventanaMatrimonios.getG();  
        String ruta = "matrimonios.txt";  
        File file = new File(ruta);  
        if (!file.exists()) {  
            file.createNewFile();  
        }  
        FileWriter fw = new FileWriter(file);  
        BufferedWriter bw = new BufferedWriter(fw);  
        g.getMatrimonios().stream().forEach(n -> {  
            try {  
                bw.write(n.toString() + "\n");  
            } catch (IOException ex) {  
                Logger.getLogger(VentanaPrincipal.class.getName()).log(Level.SEVERE, null, ex);  
            }  
        });  
        bw.close();  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

En la ventana principal creamos un objeto grupoMatrimonio donde se nos guardaran todos los matrimonios, al igual que creamos el archivo donde se guardaran dichos objetos

```

private void autoridadBtnActionPerformed(java.awt.event.ActionEvent evt) {

    int dia = Integer.parseInt(diaTxt.getText());
    int mes = Integer.parseInt(mesTxt.getText()) - 1;
    int anio = Integer.parseInt(anioTxt.getText()) - 1900;
    Date d = new Date(anio, mes, dia);
    m.agregarAutoridad(new Persona(nombreTxt.getText(), apellidoTxt.getText(), cedulaTxt.getText(), d);

}

private void contrayenteBtnActionPerformed(java.awt.event.ActionEvent evt) {

    int dia = Integer.parseInt(diaTxt.getText());
    int mes = Integer.parseInt(mesTxt.getText()) - 1;
    int anio = Integer.parseInt(anioTxt.getText()) - 1900;
    Date d = new Date(anio, mes, dia);
    m.agregarContrayente(new Persona(nombreTxt.getText(), apellidoTxt.getText(), cedulaTxt.getText(), d);

}

private void testigoBtnActionPerformed(java.awt.event.ActionEvent evt) {

    int dia = Integer.parseInt(diaTxt.getText());
    int mes = Integer.parseInt(mesTxt.getText()) - 1;
    int anio = Integer.parseInt(anioTxt.getText()) - 1900;
    Date d = new Date(anio, mes, dia);
    m.agregarTestigo(new Persona(nombreTxt.getText(), apellidoTxt.getText(), cedulaTxt.getText(), d);

}

```

En Ventana personas iremos creando cada tipo de persona (Autoridad, conyugues y testigos) para agregarlos a un matrimonio el cual luego será agregado al grupo de matrimonios


```

String rol = rolTxt.getText();
Class clase;
Object objeto;
Method metodo;
clase = Class.forName("ups.edu.ec.modelo.Matrimonio");
objeto = clase.newInstance();
metodo = clase.getMethod(
metodo.invoke(objeto, null);
} catch (ClassNotFoundException ex) {
    Logger.getLogger(VentanaAutenticacion.class.getName()).log(Level.SEVERE, null, ex);
} catch (InstantiationException ex) {
    Logger.getLogger(VentanaAutenticacion.class.getName()).log(Level.SEVERE, null, ex);
} catch (IllegalAccessException ex) {
    Logger.getLogger(VentanaAutenticacion.class.getName()).log(Level.SEVERE, null, ex);
} catch (NoSuchMethodException ex) {
    Logger.getLogger(VentanaAutenticacion.class.getName()).log(Level.SEVERE, null, ex);
} catch (SecurityException ex) {
    Logger.getLogger(VentanaAutenticacion.class.getName()).log(Level.SEVERE, null, ex);
} catch (IllegalArgumentException ex) {
    Logger.getLogger(VentanaAutenticacion.class.getName()).log(Level.SEVERE, null, ex);
} catch (InvocationTargetException ex) {
    Logger.getLogger(VentanaAutenticacion.class.getName()).log(Level.SEVERE, null, ex);
}

```

En la Ventana de autenticación, usando reflexión podremos identificar si intentamos ingresar como una autoridad, contrayente o testigo

6.
N.
RESULTADO(S) OBTENIDO(S):

 UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	Computación	Docente: Diego Quisi Peralta
	Programación Aplicada	Período Lectivo: Septiembre 2020 – Febrero 2021

CONCLUSIONES:

RECOMENDACIONES:

Nombre de estudiante: _____

Firma de estudiante: _____