	Computación	Docent: Diego Quist Peralta
	Programación Aplicada	Period Lection: September 2020 – Fiber 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 informáticos 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 jit 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, género y fecha de nacimiento) de los contrayentes. Es importante validar la equidad de género.

Igualmente se guardan 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:

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

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 Quiso Peralta MSc.

Firma: _____



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

CARRERA: Computación

ASIGNATURA: Programación Aplicada

NRO. PRÁCTICA:

1

TÍTULO PRÁCTICA: Prueba Practica 1

Desarrollo e implementación de un sistema de gestión de matrimonios de la ciudad de Cuenca

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:

El diagrama de clases se puede observar en el siguiente link:

<https://lucid.app/lucidchart/invitations/accept/0a6fcb23-07ae-4827-8522-2b0e00c5982c>

2. Patrón de diseño

El patrón de diseño aplicado en este proyecto fue el patrón singleton ya que este patrón de diseño nos ayuda a tener una instancia para todos los llamados al método, es decir que ya no sería necesario pasar como parámetro clase por clase, se aplicó a los controladores.

Patrón de diseño aplicado en controlador matrimonio:


```
public class ControladorMatrimonio extends AbstractControlador<Matrimonio>{  
  
    private static ControladorMatrimonio instance = new ControladorMatrimonio();  
  
    public static ControladorMatrimonio getInstance() {  
        return instance;  
    }  
}
```

Patrón de diseño aplicado en controlador usuario:

```
public class ControladorUsuario extends AbstractControlador<Usuario>{  
  
    private static ControladorUsuario instance = new ControladorUsuario();  
  
    public static ControladorUsuario getInstance() {  
        return instance;  
    }  
}
```

Patrón de diseño aplicado en controlador Testigo:

```
public class ControladorTestigo extends AbstractControlador<Persona>{  
  
    private static ControladorTestigo instance = new ControladorTestigo();  
  
    public static ControladorTestigo getInstane() {  
        return instance;  
    }  
}
```

	Computación	Docent: Diego Quist Peralta
	Programación Aplicada	Period Lection: September 2020 – Fiber 2021

Patrón de diseño aplicado en controlador Contrayente:

```
public class ControladorContrayente extends AbstractControlador<Persona>{

    private static ControladorContrayente instance = new ControladorContra-
yente();

    public static ControladorContrayente getInstance() {
        return instance;
    }
}
```

Patrón de diseño aplicado en controlador Juez:

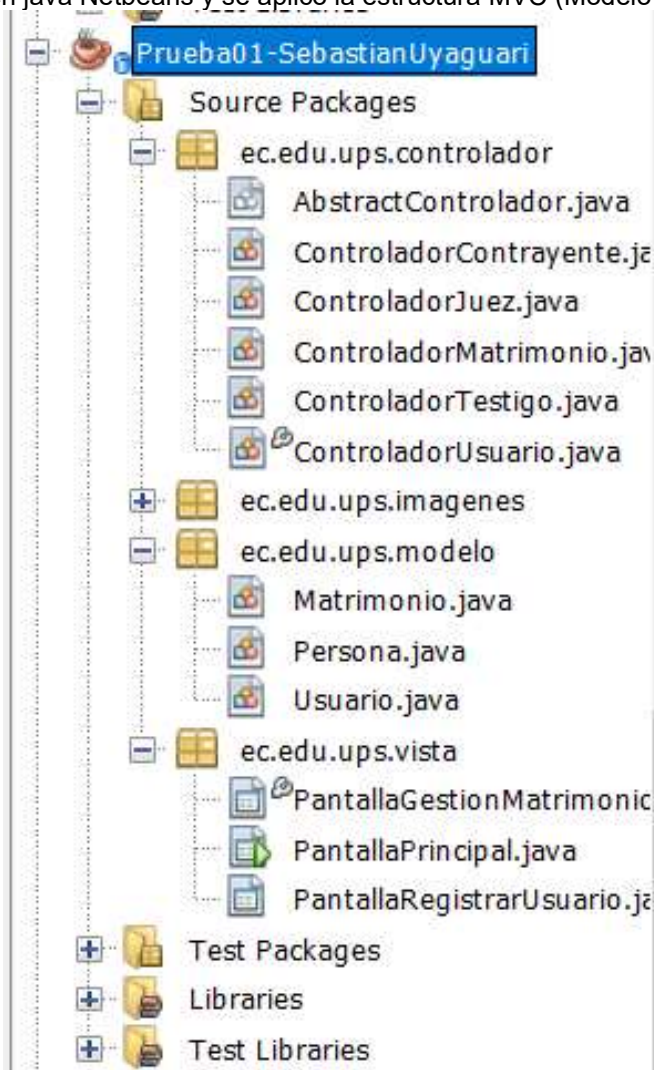
```
public class ControladorJuez extends AbstractControlador<Persona>{

    private static ControladorJuez instance = new ControladorJuez();

    public static ControladorJuez getInstance() {
        return instance;
    }
}
```

3. Descripción de la solución y pasos seguidos.

Se creó un nuevo proyecto en java Netbeans y se aplicó la estructura MVC (Modelo vista controlador):



Se utilizó la clase genérica para reducir líneas de código la cual es una clase de tipo abstracta para que las demás clases que heredan tengan sus métodos y sobrescriba otros métodos que sean específicos de la clase:

```
package ec.edu.upse.controlador;

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

/**
 *
 * @author Sebastian Uyaguari
 */
public abstract class AbstractControlador<T> {

    private List<T> lista;

    public AbstractControlador() {
        lista = new ArrayList<T>();
    }

    public boolean create(T objeto) {
        var v = validar(objeto);
        if(v==true){
            var v2 = createArchivo(objeto);
            if(v2 == true){

                return lista.add(objeto);
            }
        }
        return false;
    }

    public abstract boolean createArchivo(T objeto);

    public T read(T objeto) {
        try{
            return lista.stream().filter(t -> t.equals(objeto)).findFirst().get();
        }catch(NullPointerException ex){
            System.out.println("Error lista vacia");
        }
        return null;
    }

    public boolean update(T objeto) {
        int posicion = buscarPosicion(objeto);
        if(posicion >=0){
            if(updateArchivo(objeto)){
                lista.set(posicion, objeto);
                return true;
            }
        }
        return false;
    }

    public abstract boolean updateArchivo(T objeto);

    public boolean delite(T objeto) {
        if(lista.contains(objeto)){
            if(deliteArchivo(objeto))
                return lista.remove(objeto);
        }
        return false;
    }
}
```

```
}

public abstract boolean deliteArchivo(T objeto);

public int buscarPosicion(T buscar){
    for (int i = 0; i < lista.size(); i++) {
        Object objeto = lista.get(i);
        if(objeto.equals(buscar))
            return i;
    }
    return -1;
}

public abstract int generarCodigo();

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

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

public abstract boolean validar(T objeto);

public abstract void listar();
}
```

En los controladores se sobrescribían los métodos y se llamaba al `AccesRandomFile` para guardar la información en el archivo, Ejemplo clase `Controlador Usuario`:

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

import ec.edu.ups.modelo.Usuario;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.RandomAccessFile;
import java.util.ArrayList;
import java.util.List;

/**
 *
 * @author Sebastian Uyaguari
 */
public class ControladorUsuario extends AbstractControlador<Usuario>{

    private static ControladorUsuario instance = new ControladorUsuario();

    private static int REGISTRO;
    private RandomAccessFile archivo;

    /**
     * Estructura del archivo.
     * codigo: 4 bytes
     * usuario: 30 bytes
     * contraseña 15 bytes
     * total: 53 bytes
     */
    public ControladorUsuario() {
        try {
```

```

        REGISTRO=53;
        archivo = new RandomAccessFile("datos/usuarios.txt", "rw");
        listar();
    } catch (FileNotFoundException ex) {
        System.out.println("Error lectura/escritura |Cotrolador Usuario|");
    }
}

public static ControladorUsuario getInstance(){
    return instance;
}

@Override
public boolean createArchivo(Usuario usuario) {
    try {
        archivo.seek(archivo.length());
        archivo.writeInt(usuario.getCodigo());
        archivo.writeUTF(usuario.getUsuario());
        archivo.writeUTF(usuario.getContraseña());
        return true;
    } catch (IOException ex) {
        System.out.println("Error lectura/escritura controlador Usuario
create");
    }
    return false;
}

@Override
public boolean updateArchivo(Usuario usuario) {
    int salto = 0;
    try {
        while (salto < archivo.length()) {
            archivo.seek(salto);
            int codigo = archivo.readInt();
            if (codigo == usuario.getCodigo()) {
                archivo.seek(salto);
                archivo.writeInt(usuario.getCodigo());
                archivo.writeUTF(usuario.getUsuario());
                archivo.writeUTF(usuario.getContraseña());
                return true;
            }
            salto += REGISTRO;
        }
    } catch (IOException ex) {
        System.out.println("Error lectura/escritura controlador Usuario up-
date");
    }
    return false;
}

@Override
public boolean deliteArchivo(Usuario usuario) {
    String cadena = "";
    int salto = 0;
    try {
        while (salto < archivo.length()) {
            archivo.seek(salto);
            int codigo = archivo.readInt();
            if (codigo == usuario.getCodigo()) {
                archivo.seek(salto);
                archivo.writeInt(-100);
            }
            salto += REGISTRO;
        }
    } catch (IOException ex) {
        System.out.println("Error lectura/escritura controlador Usuario up-
date");
    }
    return false;
}

```



```

        archivo.writeUTF(String.format("%-" + 30 + "s", cadena));
        archivo.writeUTF(String.format("%-" + 15 + "s", cadena));
        return true;
    }
    salto += REGISTRO;
}
} catch (IOException ex) {
    System.out.println("Error lectura/escritura controlador Usuario de-
lite");
}
return false;
}

@Override
public int generarCodigo() {
    List<Usuario> lista = getList();
    int codigo = 0;
    if (lista.size() > 0) {
        for (Usuario usuario : lista) {
            int aux = usuario.getCodigo();
            if (aux > codigo) {
                codigo = aux;
            }
        }
        return codigo + 1;
    } else {
        return 1;
    }
}

@Override
public boolean validar(Usuario objeto) {
    return true;
}

public boolean iniciarSesion(String usuario, String contraseña) {
    List<Usuario> lista = getList();

    for (Usuario usuariol : lista) {

        if (usuariol.getUsuario().trim().equals(usuario) && usuariol.getCon-
traseña().trim().equals(contraseña)) {
            return true;
        }
    }

    return false;
}

@Override
public void listar() {
    List<Usuario> lista = new ArrayList<Usuario>();
    int salto = 0;
    try {

        while (salto < archivo.length()) {

            archivo.seek(salto);

            int codigo = archivo.readInt();
            if (codigo >= 0) {

```

```

        String usuario = archivo.readUTF();
        String contraseña = archivo.readUTF();

        Usuario u = new Usuario(codigo, usuario, contraseña);
        lista.add(u);
    }

    salto += REGISTRO;
}
setLista(lista);
} catch (IOException ex) {
    System.out.println("Error lectura/escritura controlador Usuario");
    ex.printStackTrace();
}
}
}

```

Se utilizó tres clases modelo una llamadas: persona, usuario, matrimonio:
Ejemplo clase matrimonio:

```

package ec.edu.ups.modelo;

import java.util.Date;

/**
 *
 * @author Sebastian Uyaguari
 */
public class Matrimonio {

    private int codigo;
    private String lugar;
    private Date fecha;
    private Persona contrayente1;
    private Persona contrayente2;
    private Persona testigo1;
    private Persona testigo2;
    private Persona autoridad;

    public Matrimonio() {
    }

    public Matrimonio(int codigo, String lugar, Date fecha, Persona contrayente1,
Persona contrayente2, Persona testigo1, Persona testigo2, Persona autoridad) {
        this.codigo = codigo;
        this.setLugar(lugar);
        this.fecha = fecha;
        this.contrayente1 = contrayente1;
        this.contrayente2 = contrayente2;
        this.testigo1 = testigo1;
        this.testigo2 = testigo2;
        this.autoridad = autoridad;
    }

    public int getCodigo() {
        return codigo;
    }

    public void setCodigo(int codigo) {
        this.codigo = codigo;
    }
}

```

```
}

public String getLugar() {
    return lugar;
}

public void setLugar(String lugar) {
    this.lugar = validarEspacios(lugar, 100);
}

public Date getFecha() {
    return fecha;
}

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

public Persona getContrayente1() {
    return contrayente1;
}

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

public Persona getContrayente2() {
    return contrayente2;
}

public void setContrayente2(Persona contrayente2) {
    this.contrayente2 = contrayente2;
}

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(Persona autoridad) {
    this.autoridad = autoridad;
}

public String validarEspacios(String cadena, int numero){
    if(cadena.length()==numero){
        return cadena;
    }
}
```

```

        }else{
            if(cadena.length()>numero){
                cadena = cadena.substring(0,numero);
                return cadena;
            }else{
                for (int i = cadena.length(); i < numero; i++) {
                    cadena+=" ";
                }
                return cadena;
            }
        }

    }

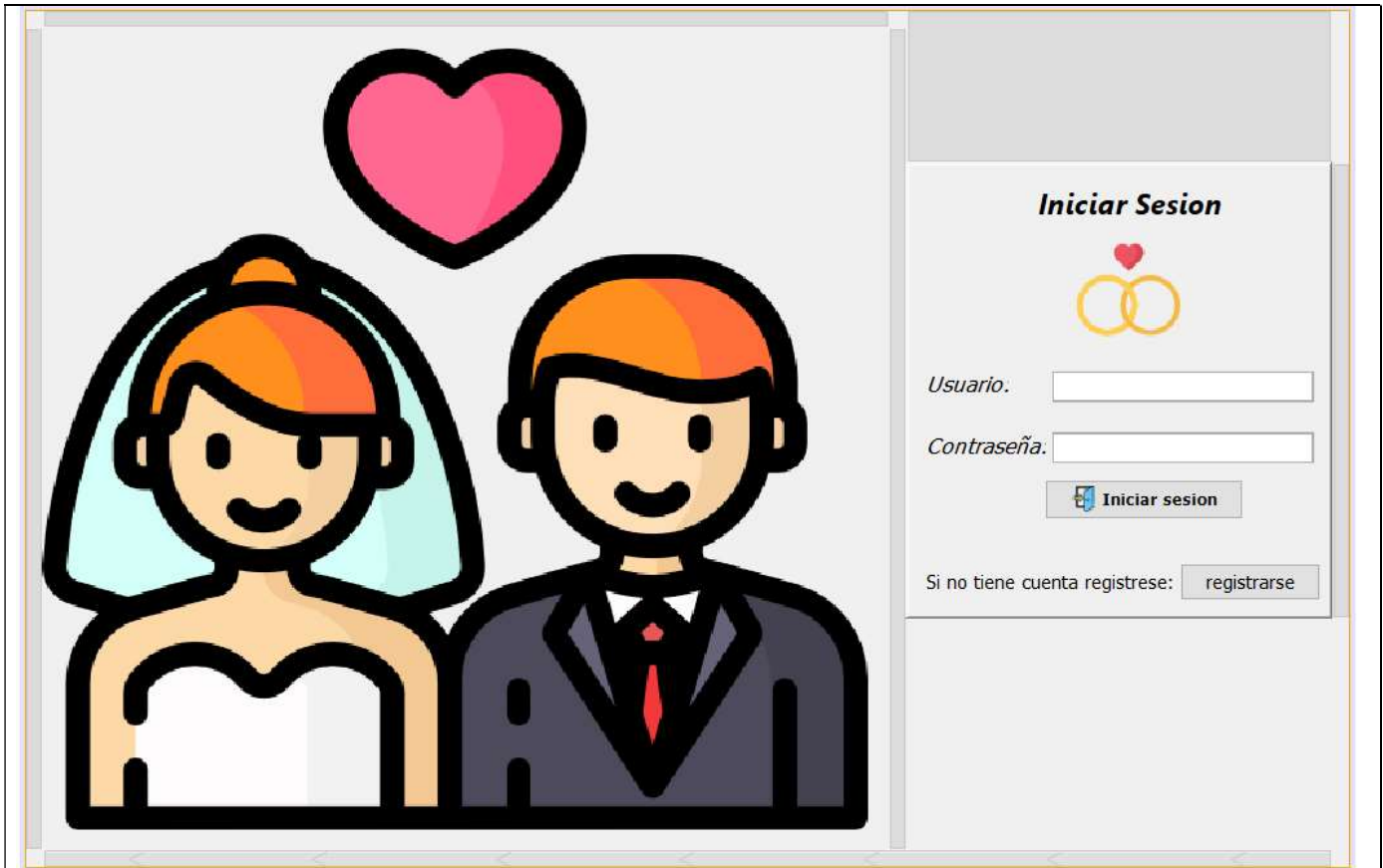
    @Override
    public int hashCode() {
        int hash = 3;
        hash = 47 * hash + this.codigo;
        return hash;
    }

    @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.codigo != other.codigo) {
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        return "Matrimonio{" + "codigo=" + codigo + ", lugar=" + lugar + ", fecha="
+ fecha + ", contrayente1=" + contrayente1 + ", contrayente2=" + contrayente2 + ",
testigo1=" + testigo1 + ", testigo2=" + testigo2 + ", autoridad=" + autoridad + '}';
    }
}

```

Se agregó vistas para interactuar con el usuario:



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.

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: Sebastián Roberto Uyaguari Ramón



Firma de estudiante: