
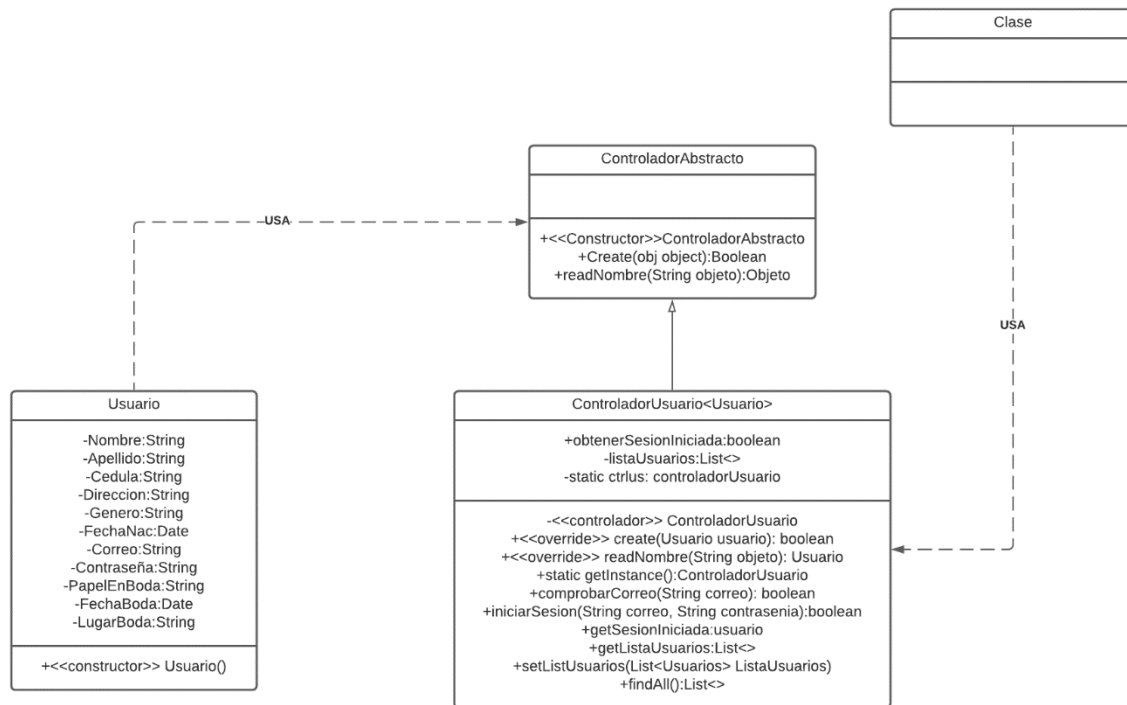
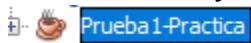
	Computación	Docente: Diego Quisi Peralta
	Programación Aplicada	Período Lectivo: septiembre 2020 – febrero 2021

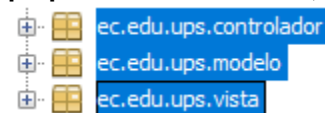
		<b>FORMATO DE GUÍA DE PRÁCTICA DE LABORATORIO / TALLERES / CENTROS DE SIMULACIÓN – PARA DOCENTES</b>	
<b>CARRERA:</b> COMPUTACIÓN/INGENIERÍA DE SISTEMAS		<b>ASIGNATURA:</b> PROGRAMACIÓN APLICADA	
<b>NRO. PROYECTO:</b>	1.1	<b>TÍTULO PRÁCTICA:</b> Prueba Practica 1 Desarrollo e implementación de un sistema de gestión de matrimonios de la ciudad de Cuenca	
<b>OBJETIVO:</b> Reforzar los conocimientos adquiridos en clase sobre la programación aplicada (Java 8, Programación Genérica, Reflexión y Patrones de Diseño) en un contexto real.			
<b>INSTRUCCIONES</b> (Detallar las instrucciones que se dará al estudiante):		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 practica en formato PDF y en conjunto con el código se debe subir al GitHub personal.	
		5. <b>Fecha de entrega:</b> El sistema debe ser subido al git hasta <b>27 de noviembre del 2020 – 23:55.</b>	
<b>ACTIVIDADES POR DESARROLLAR</b>			
<b>1. Diagrama UML</b> Para tener una guía para la implementación del código se desarrollo el diagrama de clase con todos los métodos y atributos que se usaron para la creación del programa y su debido funcionamiento, también se implementó el patrón singleton. Este es el diagrama de clases que se desarrolló:			



## 2. Creamos un Proyecto en el NetBeans con el nombre de Prueba1\_Practica

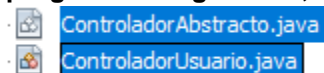


## 3. Creamos los paquetes para crear las clases y empezar con el proyecto nuevo creado, creamos los paquetes controladores, modelo y vista.

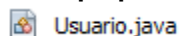


## 4. Dentro de cada paquete creamos las clases respectivas

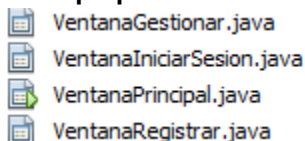
- En el paquete controlador creamos las clases controladorAbstracto que será la clase padre de las clases controladorUsuario para el usuario y también reutilizar código aplicando programación genérica, el controlador heredará los atributos y métodos de su clase padre.



- En el paquete modelo creamos la clase usuario



- En el paquete interfaz se encontrará la parte de las ventanas para dar una mejor presentación.



## 2. Creamos los atributos con sus getters y setters de la clase Usuario, también su controlador y métodos de comparación como los métodos equals y hashCode que nos serán útiles cuando busquemos datos en específico.

También el código se encuentra en el siguiente link del repositorio en GitHub:

<https://github.com/Bryambepz/Prueba1-Practica/blob/master/src/ec/edu/ups/modelo/Usuario.java>

```
public class Usuario{
    private String Nombre;
    private String Apellido;
    private String Cedula;
    private String Direccion;
    private String Genero;
    private Date FechaNac;
    private String Correo;
    private String Contraseña;
    private String PapelEnBoda;
    private String LugarBoda;
    private Date FechaBoda;

    public Usuario() {
    }

    public Usuario(String Nombre, String Apellido, String Cedula, String Direccion,
String Genero, Date FechaNac, String Correo, String Contraseña, String PapelEnBoda) {
        this.Nombre = Nombre;
        this.Apellido = Apellido;
        this.Cedula = Cedula;
        this.Direccion = Direccion;
        this.Genero = Genero;
        this.FechaNac = FechaNac;
        this.Corrreo = Correo;
        this.Contraseña = Contraseña;
        this.PapelEnBoda = PapelEnBoda;
    }

    public Usuario(String Nombre, String Apellido, String Cedula, String Direccion,
String Genero, Date FechaNac, String Correo, String Contraseña, String LugarBoda,
Date FechaBoda) {
        this.Nombre = Nombre;
        this.Apellido = Apellido;
        this.Cedula = Cedula;
        this.Direccion = Direccion;
        this.Genero = Genero;
        this.FechaNac = FechaNac;
        this.Corrreo = Correo;
        this.Contraseña = Contraseña;
        this.LugarBoda = LugarBoda;
        this.FechaBoda = FechaBoda;
    }

    public String getNombre() {
        return Nombre;
    }

    public void setNombre(String Nombre) {
        this.Nombre = Nombre;
    }

    public String getApellido() {
        return Apellido;
    }
}
```

```
public void setApellido(String Apellido) {
    this.Apellido = Apellido;
}

public String getCedula() {
    return Cedula;
}

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

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 getFechaNac() {
    return FechaNac;
}

public void setFechaNac(Date FechaNac) {
    this.FechaNac = FechaNac;
}

public String getCorreo() {
    return Correo;
}


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

public String getContraseña() {
    return Contraseña;
}

public void setContraseña(String Contraseña) {
    this.Contraseña = Contraseña;
}

public String getLugarBoda() {
    return LugarBoda;
}

public void setLugarBoda(String LugarBoda) {
    this.LugarBoda = LugarBoda;
}
```

	Computación	Docente: Diego Quisi Peralta
	Programación Aplicada	Período Lectivo: septiembre 2020 – febrero 2021

```

public Date getFechaBoda() {
    return FechaBoda;
}

public void setFechaBoda(Date FechaBoda) {
    this.FechaBoda = FechaBoda;
}

@Override
public int hashCode() {
    int hash = 3;
    hash = 71 * hash + Objects.hashCode(this.Cedula);
    hash = 71 * hash + Objects.hashCode(this.Correo);
    hash = 71 * hash + Objects.hashCode(this.Nombre);
    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 Usuario other = (Usuario) obj;
    if (!Objects.equals(this.Cedula, other.Cedula)) {
        return false;
    }
    if (!Objects.equals(this.Correo, other.Correo)) {
        return false;
    }
    if (!Objects.equals(this.Nombre, other.Nombre)) {
        return false;
    }
    return true;
}

@Override
public String toString() {
    return "Usuario{" + "Nombre=" + Nombre + ", Apellido=" + Apellido + ",
Cedula=" + Cedula + ", Direccion=" + Direccion + ", Genero=" + Genero + ", FechaNac="
+ FechaNac + ", Correo=" + Correo + ", Contrase\u00f1a=" + Contraseña + ",
PapelEnBoda=" + PapelEnBoda + ", LugarBoda=" + LugarBoda + ", FechaBoda=" + FechaBoda
+ '}';
}

}

```

3. En el paquete controlador, en la clase controladorAbstracto creamos todos los métodos abstractos que la clase hija o subclase heredara y que son necesarios para crear y leer a un usuario.

El código también se lo puede encontrar en el siguiente link:

<https://github.com/Bryambepz/Prueba1-Practica/blob/master/src/ec/edu/ups/controlador/ControladorAbstracto.java>

```
public abstract class ControladorAbstracto<T> {

    public ControladorAbstracto() {
    }

    public abstract boolean create(T objeto);

    public abstract T readNombre(String objeto);

}
```

En la clase controladorUsuario se encontrarán los métodos que se heredaran de la clase padre controladorAbstracto, a esta clase se le aplico el patrón singleton, y esta es a la que se le llama para crear al usuario, buscar un usuario en específico o comprobar el Login del usuario.

El código se encuentra en el siguiente link:

<https://github.com/Bryambepz/Prueba1-Practica/blob/master/src/ec/edu/ups/controlador/ControladorUsuario.java>

```
public class ControladorUsuario extends ControladorAbstracto<Usuario> {

    private List<Usuario> listaUsuarios;
    // private List<List> listaCasados;
    private static ControladorUsuario ctrlus;

    private ControladorUsuario() {
        listaUsuarios = new ArrayList<>();
    }
    // listaCasados = new ArrayList<>();

    public static ControladorUsuario getInstance() {
        if (ctrlus == null) {
            ctrlus = new ControladorUsuario();
        }
        return ctrlus;
    }

    @Override
    public boolean create(Usuario objeto) {
        if (!listaUsuarios.contains(objeto)) {
            return listaUsuarios.add(objeto);
        }
        return true;
    }

    // public boolean createCasados(Usuario objeto, Usuario objeto2, Usuario objeto3,
    Usuario objeto4) {
    // if (listaCasados.contains(objeto)) {
    // List<Usuario> c = new ArrayList<>();
    // c.add(objeto);
    // c.add(objeto2);
    // c.add(objeto3);
    // c.add(objeto4);
    // return listaCasados.add(c);
    // }
}
```

```

////      return true;
//      }

@Override
public Usuario readNombre(String objeto) {
    for (int i = 0; i < listaUsuarios.size(); i++) {
        var us = listaUsuarios.get(i);
        if (objeto.equals(us.getNombre())) {
            return us;
        }
    }
    return null;
}

public boolean comprobarCorreo(String correo) {
    return listaUsuarios.stream().filter(c ->
correo.equals(c.getCorreo())).noneMatch(c -> correo.equals(c.getCorreo()));
}

Usuario obtenerSesionIniciada;

public boolean iniciarSesion(String correo, String contrasenia) {
    for (int i = 0; i < listaUsuarios.size(); i++) {
        var us = listaUsuarios.get(i);
        if (correo.equals(us.getCorreo()) &&
contrasenia.equals(us.getContraseña())) {
            obtenerSesionIniciada = us;
            return true;
        }
    }
    return false;
}

public Usuario getSessionIniciada() {
    return obtenerSesionIniciada;
}

public List<Usuario> getListUsuarios() {
    return listaUsuarios;
}

public void setListUsuarios(List<Usuario> listaUsuarios) {
    this.listaUsuarios = listaUsuarios;
}

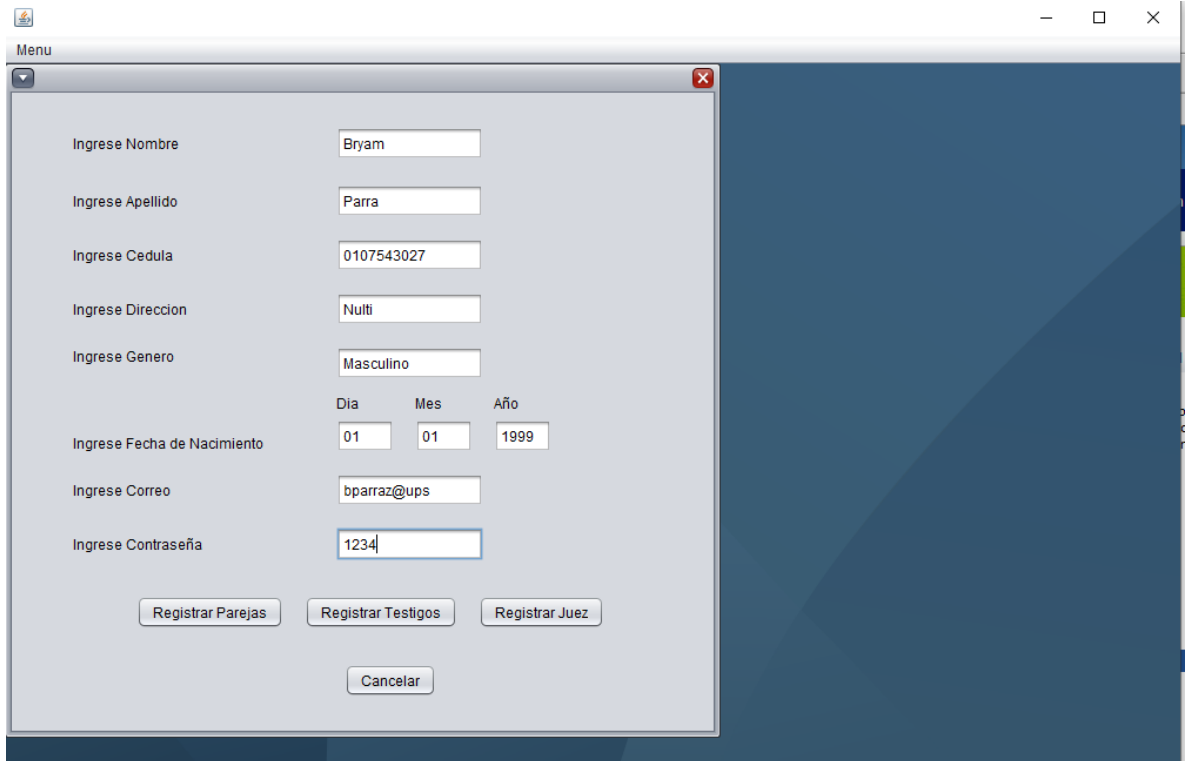
public List<Usuario> findAll() {
    return listaUsuarios;
}

//      public List<List> findAll2() {
//          return listaCasados;
//      }
}

```

## 5. Funcionamiento de la aplicación

- a. Registramos a un usuario nuevo, al crear comprobamos si el correo ingresado no existe y así crear al nuevo usuario, aquí se podrá registrar uno mismo, la pareja, los 2 testigos y el juez.

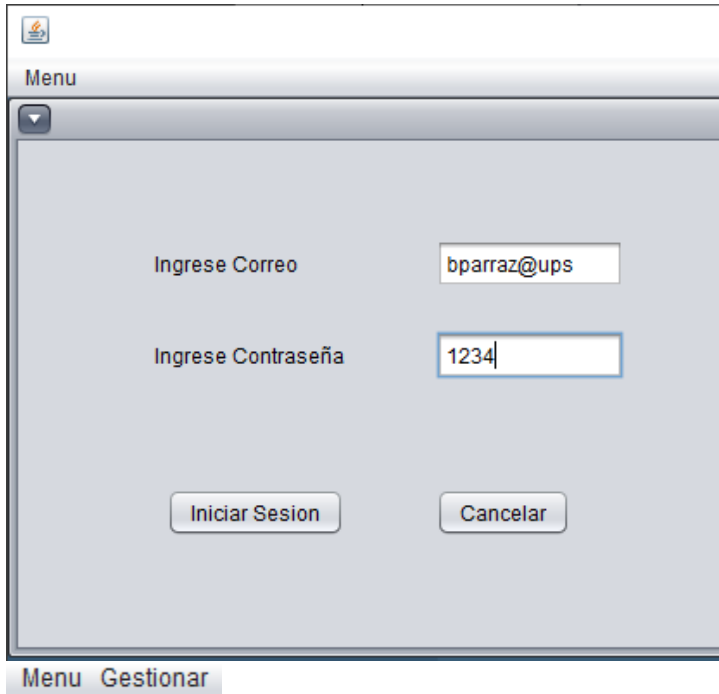


The screenshot shows a registration window titled "Menu" with a dropdown arrow. The form contains the following fields and buttons:

- Ingrese Nombre:
- Ingrese Apellido:
- Ingrese Cedula:
- Ingrese Direccion:
- Ingrese Genero:
- Ingrese Fecha de Nacimiento: 

Dia	Mes	Año
<input type="text" value="01"/>	<input type="text" value="01"/>	<input type="text" value="1999"/>
- Ingrese Correo:
- Ingrese Contraseña:
- Buttons: , , ,

- b. Para iniciar sesión y gestionar el matrimonio del usuario, verificamos si el correo y la contraseña son correctos y podrá ingresar a gestionar, caso contrario no podrá acceder al mismo. Si los datos de Login son correctos se visualizará un nuevo menú llamado gestionar.




The screenshot shows the login form and the resulting "Menu Gestionar" window. The login form has the following fields and buttons:

- Ingrese Correo:
- Ingrese Contraseña:
- Buttons: ,

Below the login form, a new window titled "Menu Gestionar" is shown, indicating a successful login.

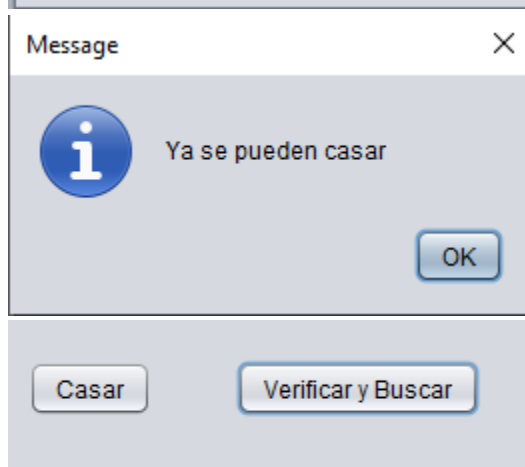


- c. En el gestor podrá ingresar el lugar donde se realizará el matrimonio y la fecha, en los otros campos se ingresará el nombre de la pareja, los 2 testigos y del juez, para asegurarse de que esas personas se encuentran registradas el botón verificar y buscar, analizará si los datos son correctos y existe caso contrario el botón casarse no estará habilitado hasta que registre a las que formalizan el acto.



The screenshot shows a web form for marriage registration. It includes the following fields and buttons:

- Lugar de la Ceremonia:** A dropdown menu with "Registro Civil" selected.
- Fecha de la Ceremonia:** Three input fields for day, month, and year, containing "1", "01", and "2021" respectively.
- Pareja:** A text input field containing "Paula".
- Testigo 1:** A text input field containing "omar".
- Testigo 2:** A text input field containing "Rosario".
- Juez:** A text input field containing "Hernan".
- Buttons:** "Casar" (disabled) and "Verificar y Buscar" (active).



The screenshot shows a message dialog box with the following elements:

- Title:** "Message" with a close button (X).
- Icon:** A blue circle with a white lowercase "i" (information icon).
- Text:** "Ya se pueden casar".
- Buttons:** "OK" and "Casar" (disabled).

- d. Por último presione en el botón "Casar" que se habilita y se registrará su matrimonio.

**RESULTADO(S) OBTENIDO(S):**

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

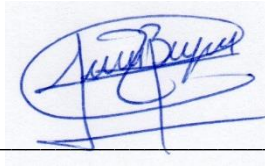
**CONCLUSIONES:**

- Los estudiantes identifican las principales estructuras para la creación de sistemas informáticos.
- Los estudiantes implementan soluciones graficas 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:** Bryam Parra



**Firma de estudiante:** \_\_\_\_\_